(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2002－49585
（P2002－49585A）
(43)公開日　平成14年２月15日(2002.2.15)

(51)Int.Cl.[7]	識別記号	ＦＩ		テーマコード*（参考）
Ｇ０６Ｆ　15/00	３１０	Ｃ０６Ｆ　15/00	３１０Ａ	５Ｂ０７５
9/46	３６０	9/46	３６０Ｂ	５Ｂ０８５
13/00	５６０	13/00	５６０Ａ	５Ｂ０９８
17/30	１１０	17/30	１１０Ｃ	
			１１０Ｆ	

審査請求　未請求　請求項の数24　ＯＬ　外国語出願　（全 76 頁）　最終頁に続く

(21)出願番号　特願2001－129922(P2001－129922)

(22)出願日　平成13年４月26日(2001.4.26)

(31)優先権主張番号　０９／５７１１４４

(32)優先日　平成12年５月18日(2000.5.18)

(33)優先権主張国　米国（ＵＳ）

(71)出願人　595124929
マイクロソフト　コーポレイション
ＭＩＣＲＯＳＯＦＴ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国　98052－6399　ワシントン州　レドマンド　ワン　マイクロソフト　ウェイ　（番地なし）

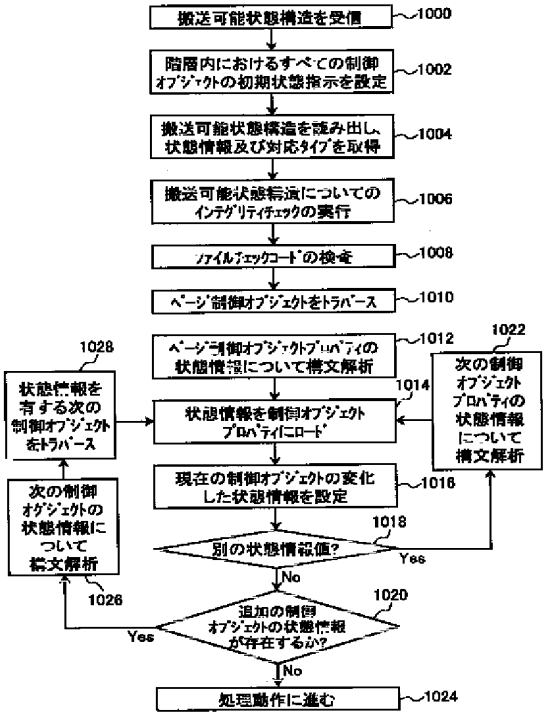
(72)発明者　バード、ゲイリー　エス.
アメリカ合衆国、98033　ワシントン州、カークランド、エヌイー　103ド　ストリート　11411

(74)代理人　100095555
弁理士　池内　寛幸　（外３名）

最終頁に続く

(54)【発明の名称】　サーバ側制御オブジェクトの状態管理方法

(57)【要約】　（修正有）
【課題】　最小のプログラムでウェブページを動的に作成および処理できる開発フレームワークを提供する。
【解決手段】　ウェブサーバとクライアントとの間を往復する搬送可能状態構造を用いて１つ以上のサーバ側制御オブジェクトの状態を管理する。搬送可能状態構造の内、状態情報はサーバ側制御オブジェクトのプロパティに関し、階層情報は関連する状態情報をロードすべき階層内の適切なサーバ側制御オブジェクトを探すためにサーバによって用いられ、さらにインテグリティコードは、サーバとクライアントの間の往復の間にサーバが搬送可能状態構造が変造されたかどうかをサーバが判断するのを可能にする。クライアントへのレスポンス前に、1つ以上のサーバ側制御オブジェクトの状態情報をクライアントへ送信する搬送可能状態構造に記録する。そして、搬送可能状態構造は、サーバに戻され、サーバ側制御オブジェクトにロードされ、階層を以前の状態に戻す。



【特許請求の範囲】
【請求項１】　クライアントに表示されるウェブページに組み込まれるクライアント側ユーザインタフェース要素に対応するサーバ側制御オブジェクトの状態管理方法であって、
クライアント側ユーザインタフェース要素を処理するためのサーバ側制御オブジェクトを制御オブジェクト階層に作成する工程と、
前記制御オブジェクト階層における少なくとも１つの前記サーバ側制御オブジェクトの状態値を示す状態情報を含む搬送可能状態構造をクライアントから受信する工程と、
前記搬送可能状態構造から前記状態情報を抽出する工程と、
前記状態情報からの状態値がサーバ側制御オブジェクトに関連している場合、その状態値をサーバ側制御オブジェクトのプロパティにロードする工程とを含むサーバ側制御オブジェクトの状態管理方法。
【請求項２】　前記抽出する工程において、
前記搬送可能状態構造から前記少なくとも１つの前記サーバ側制御オブジェクトのプロパティに関連する状態値を抽出する工程と、
前記搬送可能状態構造内の階層情報に基づき前記制御オブジェクト階層内の前記サーバ側制御オブジェクトの階層位置を認識する工程とを含む請求項１に記載のサーバ側制御オブジェクトの状態管理方法。
【請求項３】　前記ロードする工程において、
識別処理によって識別された階層位置に基づき前記制御オブジェクト階層内の前記サーバ側制御オブジェクトを指示する工程と、
前記制御オブジェクト階層内の前記サーバ側制御オブジェクトの階層位置をトラバースする工程と、
前記サーバ側制御オブジェクトのプロパティに状態値を格納する工程とを含む請求項２に記載のサーバ側制御オブジェクトの状態管理方法。
【請求項４】　前記搬送可能状態構造は、受信インテグリティコードをさらに含み、さらに、
前記搬送可能状態構造から前記受信インテグリティコードを読み出す工程と、
前記搬送可能状態構造に含まれる状態情報から計算インテグリティコードを計算する工程と、
前記受信インテグリティコードと前記計算インテグリティコードを比較し、前記搬送可能状態構造が有効かどうかを判断する工程とを含むサーバ側制御オブジェクトの状態管理方法。
【請求項５】　前記サーバ側制御オブジェクトのプロパティが初期状態となるように初期化する工程と、
前記サーバ側制御オブジェクトのプロパティの状態が前記初期状態から変化していないことを示す指標を設定する工程と、
プロパティに前記搬送可能状態構造からの状態値がロードされた場合、前記サーバ側制御オブジェクトのプロパティの状態が前記初期状態から変化したことを示すように前記指標を変更する工程とを含む請求項１に記載のサーバ側制御オブジェクトの状態管理方法。
【請求項６】　前記プロパティがサーバ側データ格納フィールドに結合されたデータである場合、前記サーバ側制御オブジェクトのプロパティが、初期状態から変化していないことを示す指標を設定する工程をさらに含む請求項５に記載のサーバ側制御オブジェクトの状態管理方法。
【請求項７】　前記制御オブジェクト階層内における前記各サーバ側制御オブジェクトをトラバースする工程と、
前記指標が、前記サーバ側制御オブジェクトの状態情報が初期状態から変化したことを表している場合、クライアントへ送り返すための前記搬送可能状態構造に前記サーバ側制御オブジェクトの状態情報を格納する工程とをさらに含む請求項５に記載のサーバ側制御オブジェクトの状態管理方法。
【請求項８】　ウェブページを定義するオーサリング言語データを有するレスポンスによって、クライアントに前記搬送可能状態構造を送信する工程をさらに含む請求項７に記載のサーバ側制御オブジェクトの状態管理方法。
【請求項９】　前記状態情報を格納する工程に応答して、前記搬送可能状態構造に含まれる状態情報からインテグリティコードを計算する工程と、
前記搬送可能状態構造に前記インテグリティコードを格納する工程と、
ウェブページを定義するオーサリング言語データを有するレスポンスによって、クライアントに前記搬送可能状態構造を送信する工程とをさらに含む請求項７に記載のサーバ側制御オブジェクトの状態管理方法。
【請求項１０】　前記制御オブジェクト階層の少なくとも１つの前記サーバ側制御オブジェクトからウェブページの少なくとも一部を定義するオーサリングコードを生成する工程をさらに含む請求項１に記載のサーバ側制御オブジェクトの状態管理方法。
【請求項１１】　オーサリング言語コードによって定義され、ウェブサーバに接続されたクライアントコンピュータシステム上で実行処理するブラウザによって解釈可能なウェブページであって、
前記オーサリング言語コードは、実質的に請求項１０に記載のサーバ側制御オブジェクトの状態管理方法によってウェブサーバで生成され、
前記オーサリング言語コードは、前記制御オブジェクト階層の少なくとも１つの前記サーバ側制御オブジェクトの状態情報を格納する前記搬送可能状態構造を含むウェブページ。

【請求項１２】　コンピュータシステムによって読み取り可能であり、クライアントに表示されるウェブページに組み込まれるクライアント側ユーザインタフェース要素に対応するサーバ側制御オブジェクトの状態を管理するコンピュータプロセスを実行処理するコンピュータプログラムをコード化するコンピュータプログラム記憶媒体であって、
前記コンピュータプロセスは、
制御オブジェクト階層に前記サーバ側制御オブジェクトを作成して前記クライアント側ユーザインタフェース要素を処理する工程と、
前記制御オブジェクト階層の少なくとも１つの前記サーバ側制御オブジェクトの状態値を示す状態情報を含む搬送可能状態構造をクライアントから受信する工程と、
前記搬送可能状態構造から状態情報を抽出する工程と、
前記状態情報からの状態値が前記サーバ側制御オブジェクトと関連している場合、その状態値を前記サーバ側制御オブジェクトのプロパティにロードする工程とを含むコンピュータプログラム記憶媒体。
【請求項１３】　コンピュータシステムによって搬送波で具体化され、クライアントに表示されるウェブページに組み込まれるクライアント側ユーザインタフェース要素に対応するサーバ側制御オブジェクトの状態を管理するコンピュータプロセスを実行するコンピュータプログラムをコード化するコンピュータデータ信号であって、
前記コンピュータプロセスは、
制御オブジェクト階層に前記サーバ側制御オブジェクトを作成して前記クライアント側ユーザインタフェース要素を処理する工程と、
前記制御オブジェクト階層の少なくとも１つの前記サーバ側制御オブジェクトの状態値を示す状態情報を含む搬送可能状態構造をクライアントから受信する工程と、
前記搬送可能状態構造から状態情報を抽出する工程と、
前記状態情報からの状態値が前記サーバ側制御オブジェクトと関連している場合、その状態値を前記サーバ側制御オブジェクトのプロパティにロードする工程とを含むコンピュータデータ信号。
【請求項１４】　制御オブジェクト階層に作成され、クライアントに表示されるウェブページに組み込まれる複数のクライアント側ユーザインタフェース要素に対応する複数のサーバ側制御オブジェクトの状態を管理するコンピュータプロセスをコンピュータシステムにおいて実行するコンピュータプログラムをコード化するコンピュータプログラムプロダクトであって、
前記コンピュータプロセスは、
前記制御オブジェクト階層の１つ以上の前記サーバ側制御オブジェクトに関連する状態情報を含む搬送可能状態構造をクライアントから受信する工程と、
前記状態情報を非直列化して状態値、関連プロパティデータタイプおよび前記サーバ側制御オブジェクトのプロパティの階層情報を抽出する工程と、
前記階層情報に基づき前記制御オブジェクト階層内の前記サーバ側制御オブジェクトを探し当てる工程と、
前記状態値を前記サーバ側制御オブジェクトのプロパティにロードする工程とを含むコンピュータプログラムプロダクト。
【請求項１５】　前記コンピュータプロセスは、前記ロードする工程以前に前記関連プロパティデータタイプを有するように状態値を変換する工程をさらに含む請求項１４に記載のコンピュータプログラムプロダクト。
【請求項１６】　前記コンピュータプロセスは、前記サーバ側制御オブジェクトのプロパティを初期状態となるように初期化する工程と、
前記サーバ側制御オブジェクトのプロパティが前記初期状態から変化していないことを示す指標を設定する工程と、
前記プロパティに前記搬送可能状態構造からの状態値がロードされた場合、前記サーバ側制御オブジェクトのプロパティの状態が初期状態から変化したことを示すように前記指標を変更する工程とを含む請求項１４に記載のコンピュータプログラムプロダクト。
【請求項１７】　前記ロードする工程が、前記制御オブジェクト階層の1つ以上の前記サーバ側制御オブジェクトをトラバースする工程を含み、
前記各サーバ側制御オブジェクトに対し、前記ロードする工程は、
前記サーバ側制御オブジェクトの状態が初期状態から変化したことを示す指標と関連する前記サーバ側制御オブジェクトからプロパティ値を抽出する工程と、
前記プロパティ値を、前記制御オブジェクト階層内の他の前記サーバ側制御オブジェクトからのプロパティ値を有する、クライアントへ送り返すための前記搬送可能状態構造に直列化する工程とをさらに含むコンピュータプログラムプロダクト。
【請求項１８】　前記ロードする工程が、前記プロパティ値と関連する前記サーバ側制御オブジェクトからプロパティデータタイプを抽出する工程と、
前記プロパティデータタイプを、前記サーバ側制御オブジェクトのプロパティ値を有する、クライアントへ送り返すための前記搬送可能状態構造に直列化する工程とをさらに含む請求項１６に記載のコンピュータプログラムプロダクト。
【請求項１９】　前記ロードする工程が、クライアントへ送り返すための前記搬送可能状態構造に、前記制御オブジェクト階層の前記サーバ側制御オブジェクトに関する階層情報を格納する工程をさらに含む請求項１６に記載のコンピュータプログラムプロダクト。
【請求項２０】　前記コンピュータプロセスが、ウェブページを定義するオーサリング言語データを有するレスポンスによってクライアントに前記搬送可能状態構造を

送信する工程をさらに含む請求項１７に記載のコンピュータプログラムプロダクト。
【請求項２１】　前記コンピュータプロセスが、前記直列化する工程に応答して、前記搬送可能状態構造に格納された状態情報からインテグリティコードを計算する工程と、
前記搬送可能状態構造に前記インテグリティコードを格納する工程と、
ウェブページを定義するオーサリング言語データを有するレスポンスによって前記搬送可能状態構造をクライアントに送信する工程とをさらに含む請求項１７に記載のコンピュータプログラムプロダクト。
【請求項２２】　前記搬送可能状態構造は、受信インテグリティコードをさらに含み、さらに、
前記搬送可能状態構造から前記受信インテグリティコードを読み出す工程と、
前記搬送可能状態構造に含まれる状態情報から計算インテグリティコードを計算する工程と、
前記受信インテグリティコードと前記計算インテグリティコードを比較し、前記搬送可能状態構造が有効かどうかを判断する工程とを含むコンピュータプログラムプロダクト。
【請求項２３】　クライアントコンピュータからの状態値およびデータタイプを格納する搬送可能状態構造を含むリクエストを処理するサーバコンピュータで実行処理されるハンドラシステムであり、
ハンドラによって作成され、クライアント側ユーザインタフェース要素と関連するサーバ側制御オブジェクトの階層であって、少なくとも１つのサーバ側制御オブジェクトがプロパティを含むサーバ側制御オブジェクトの階層と、
データタイプに基づき、前記搬送可能状態構造に受け取った状態値を前記サーバ側制御オブジェクトのプロパティに格納するローダモジュールと、
前記搬送可能状態構造に前記プロパティおよびデータタイプを格納する保存モジュールとを含むハンドラシステム。
【請求項２４】　状態値、関連階層情報および搬送可能状態構造からのデータタイプを抽出する読み出しモジュールと、
前記関連階層情報に基づき前記制御オブジェクト階層内で制御オブジェクトを指示するトラバースモジュールとをさらに含む請求項２３に記載のハンドラシステム。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、一般にウェブサーバフレームワークに関し、特にウェブページのクライアント側ユーザインタフェース要素を処理するサーバ側制御オブジェクトの状態の管理に関する。
【０００２】
【従来の技術】典型的なウェブブラウザは、クライアントシステムにおいて表示するウェブページの外観および基本動作を規定するウェブサーバからデータを受け取る。典型的な手順としては、ユーザが、ワールドワイドウェブ上の資源のグローバルアドレスであるユニホームリソース（資源）ロケータ（以下、「URL」という）を特定し、所望のウェブサイトにアクセスする。一般に、用語「資源」とは、プログラムによってアクセスすることができるデータまたはルーチンのことである。URLの一例は、“ HYPERLINK "http://www.microsoft.com/ms.htm" http://www.microsoft.com/ms.htm ”である。このURL例の第１の部分は、通信に用いる所与のプロトコル（例えば“http”）を示している。第２の部分は、資源の所在を示すドメイン名（例えば“ HYPERLINK "http://www.microsoft.com" www.microsoft.com ”）を特定している。第３の部分はドメイン内の資源（例えば“ms.htm”と呼ばれるファイル）を特定している。これに従い、ブラウザは、 HYPERLINK "http://www.microsoft.com" www.microsoft.com ドメイン内のms.htmファイルに関連するデータを取り出すためのURL例に関連するHTTP(ハイパーテキストトランスポートプロトコル)リクエストを生成する。www.microsoft.comサイトをホストしているウェブサーバはHTTPリクエストを受け取り、要求されたウェブページまたは資源をクライアントシステムにHTTPレスポンスで戻し、ブラウザに表示する。
【０００３】上記の例の"ms.htm"ファイルは、静的なHTML（ハイパーテキストマークアップ言語）コードを含んでいる。HTMLは、ワールドワイドウェブ上でのドキュメント（例えば、ウェブページ）作成に用いられるプレイン（平文）テキストオーサリング言語である。このようなものであるので、HTMLファイルは、ウェブサーバから取り出され、ブラウザにウェブページとして表示することができる。HTMLを用いて、開発者が、例えば、ブラウザに表示するフォーマット化されたテキスト、リスト、フォーム、テーブル ハイパーテキストリンク、インライン画像および音声、ならびに背景グラフィックスを特定し、ユーザがインターネットからの情報を見ている間に期待するようになるグラフィカルな経験を提供することができる。しかし、HTMLファイルは、ウェブページコンテンツの動的な生成を本質的にサポートしていない静的ファイルである。
【０００４】株価の変化や交通情報などのような動的コンテンツを表示する場合は、通常、より複雑なクライアント・サーバ間の対話をハンドリングするためのサーバ側アプリケーションプログラムを開発する。サーバ側アプリケーションプログラムは、HTTPリクエストを処理し、クライアントへのHTTPレスポンスの送信に適切なHTMLコードを生成する。HTTPリクエストの一例は、照会ストリングでのデータまたはウェブベースのフォームからのデータなどのパラメータを含むことがある。このよう

なものであるので、サーバ側アプリケーションプログラムは、このパラメータを処理し、クライアントへのHTTPレスポンスでHTMLコードを動的に生成することができる。サーバ側アプリケーションプログラムの一例として、メモリ機構への１つ以上のフォーマット化されたテキスト書込み処理のシーケンスを用いて、適切なHTMLコードを含むドキュメントを動的に生成する場合がある。その後、得られたドキュメントは、HTTPレスポンスでクライアントシステムに送信され、そこでウェブページとしてブラウザに表示される。

【０００５】サーバ側アプリケーションプログラムの開発は、ウェブページ設計に用いる通常のHTMLコード化に精通しているだけでなく、１つ以上のプログラム言語（例えば、C++, Perl, Visual Basic, または Jscript）を含むプログラムベーシックスに精通していることが要求される複雑な作業である。しかし、ウェブページ設計者は、グラフィックデザイナーまたはエディタであることが多く、プログラム経験がない場合がある。さらに、複雑なウェブページ開発を単純化すると、いかなる開発者によっても新たなウェブコンテンツの開発の速度を上げることができる。一般に、カスタムサーバ側アプリケーションプログラムの開発もまた、多大な労力が要求され、実際、開発者はしばしばそれを試みたくないと思う程である。したがって、開発者が最小のプログラムでウェブページを動的に作成および処理することができる開発フレームワークを提供することが望ましい。

【０００６】

【発明が解決しようとする課題】動的ウェブページ生成のプログラム要件を最小にする１つの手段は、マイクロソフト社によって提供されるアクティブサーバページ（ASP）フレームワークである。ASPリソースは、典型的に、所望の資源としてASPリソースを特定するHTTPリクエストを処理し、その後、クライアントへのHTTPレスポンスでの結果HTMLコードを生成する、例えばVisual BasicまたはJscriptを含む。さらに、ASP資源は、ページ開発者がすべてのコンポーネントをスクラッチから（何もない状態から）書込むことを要求するのではなくて、予め開発されたまたは第三者のクライアント側ライブラリコンポーネント（例えば、クライアント側”ACTIVEX”制御）を参照することができる。しかし、現在のサーバ側アプリケーションフレームワークにおいて、サーバ側アプリケーション内でクライアント側ユーザインタフェース要素（例えば、テキストボックス、リストボックス、ボタン、ハイパーリンク、画像、音声など）を動的に管理しなければならないプログラミングは、依然として高度なプログラミング技術と相当な労力を必要とする。未解決の課題は、ウェブページ開発者がウェブページの他のアスペクトに焦点を当てることができるように、ユーザインタフェース要素を処理することが要求されるプロパティカプセル化プログラミングについてである。

【０００７】しかし、クライアント側ユーザインタフェース要素のサーバ側処理は、典型的にサーバ側アプリケーションプログラムでの複雑な状態管理問題を伴うことがある。クライアント側ユーザインタフェース要素に対応するサーバ側処理モジュールの状態は、サーバ側モジュールのプロパティおよび構成を表すものである。しかし、クライアントとサーバ間の通信接続は予期せず切断されることがあるので、状態管理を伴わないクライアント／サーバモデルでは、クライアントとサーバとの間の通信において、サーバがクライアントとサーバ間に発生するプロセスの状態を維持しないよう指示されてしまう。

【０００８】

【課題を解決するための手段】本発明によると、上述および他の課題は、サーバが２つのリクエストの間における状態を保持していないクライアント側ユーザインタフェース要素について、サーバ側処理のカプセル化された状態管理を提供することによって解決される。クライアント側ユーザインタフェース要素の処理は、状態管理処理を含み、これは、制御オブジェクト階層の１つ以上のサーバ側制御オブジェクトの状態（例えば、「ビューステート」（viewstate））に関するものである。サーバ側制御オブジェクトはウェブページに表示するクライアント側ユーザインタフェース要素を処理および生成する。サーバ側制御オブジェクトの階層はまた、クライアントのウェブページに表示する標準HTMLのような結果オーサリング言語コードを協働して生成する。クライアントは、例えば、標準HTMLまたは別のオーサリング言語をサポートするものであれば、いかなるブラウザであってもよい。

【０００９】状態管理を行わないクライアント／サーバモデルにおいては（を満たすために）、サーバ側制御オブジェクトの状態情報は、２つのHTTPリクエストの間はサーバ側に保持されるのではなく、搬送可能な状態構造でクライアントとサーバ間を搬送されることになる。サーバがクライアントからHTTPリクエストを受信すると、サーバは、HTTPリクエストからの搬送可能状態構造から状態情報を抽出し、この状態情報をサーバ側階層の適切な個々の制御オブジェクトに分配する。搬送可能状態構造のインテグリティもまた、搬送可能状態構造に関連し、かつ状態情報から生成されたコード化された構造を用いて検査される。

【００１０】クライアントに表示されるウェブページに組み込まれるクライアント側ユーザインタフェース要素に対応するサーバ側制御オブジェクトの状態管理方法を提供する。制御オブジェクト階層にサーバ側制御オブジェクトを作成して、クライアント側ユーザインタフェース要素を処理する。搬送可能状態構造をクライアントから受け取る。搬送可能状態構造は、制御オブジェクト階層の少なくとも１つのサーバ側制御オブジェクトの状態

値を示す状態情報を含む。状態情報は搬送可能状態構造から抽出される。状態情報からの状態値がサーバ側制御オブジェクトに関連している場合、その状態値をサーバ側制御オブジェクトのプロパティにロードする。

【００１１】制御オブジェクト階層に作成され、かつクライアントに表示されるウェブページに組み込まれる複数のクライアント側ユーザインタフェース要素に対応する複数のサーバ側制御オブジェクトの状態を管理するコンピュータプログラムプロダクトを提供する。クライアントから受信した搬送可能状態構造は、制御オブジェクト階層の１つ以上のサーバ側制御オブジェクトに関連する状態情報を含む。状態情報を直列化して、状態値、関連したプロパティデータタイプおよびサーバ側制御オブジェクトのプロパティの階層情報が抽出される。階層情報に基づき制御オブジェクト階層内のサーバ側制御オブジェクトを探し当てる。状態値をサーバ側制御オブジェクトのプロパティにロードする。

【００１２】コンピュータプログラムプロダクトとしての製品を提供する。本発明によるコンピュータプログラムプロダクトのある実施形態は、コンピュータシステムによって読み取り可能であり、クライアントに表示されるウェブページに組み込まれるクライアント側ユーザインタフェース要素に対応するサーバ側制御オブジェクトの状態を管理するコンピュータプロセスを実行するコンピュータプログラムをコード化するコンピュータプログラム記憶媒体を含む。本発明によるコンピュータプログラムプロダクトの別の実施形態は、コンピュータシステムによって搬送波で具体化され、クライアントに表示されるウェブページに組み込まれるクライアント側ユーザインタフェース要素に対応するサーバ側制御オブジェクトの状態を管理するコンピュータプログラムをコード化するコンピュータデータ信号を含む。本発明のプロセスによって生成されたプロダクトは、クライアントに送られクライアント上のブラウザで解釈可能な搬送可能状態構造を含むオーサリング言語コードとして提供される。

【００１３】

【発明の実施の形態】本発明のある実施形態において、ウェブページコンテンツは、ウェブサーバ上に動的に生成され、クライアントに表示される。クライアントとウェブサーバは、例えば、HTTPリクエストおよびHTTPレスポンスを用いてネットワークにより通信する。このように、ウェブサーバは、HTMLコード形態などのウェブページコンテンツを生成し、ブラウザにウェブページを表示することができるクライアントへそのコンテンツを送信する。ウェブページの個々のユーザインタフェース要素と論理的に対応するサーバ側制御オブジェクトが、ウェブサーバ上に作成され、ウェブページを表示および処理するクライアント側ブラウザによって用いられるウェブページコンテンツを処理および生成する。サーバ側制御オブジェクトは、サーバ側制御オブジェクトの階層を作成するページファクトリによって処理されるASP+リソースなどの動的コンテンツファイルの中で宣言される。階層における制御オブジェクトはクライアントから受信したリクエストを協働して処理し、次いで、結果ウェブページコンテンツを生成し、クライアントへ送信し、階層における制御オブジェクトを終了させる。

【００１４】本発明のある実施形態において、ページオブジェクトは、制御オブジェクト階層の最上レベルとして例示することができる。制御オブジェクトでもあるページオブジェクトは、典型的に１つ以上の子制御オブジェクトを含み、各子制御オブジェクトは、それ自体の１つ以上の子制御オブジェクトを含み、多数レベルの階層に広がる。ページオブジェクトおよび子孫の制御オブジェクトは、処理シーケンスを実行し、クライアント側ユーザインタフェース要素に対応するウェブコンテンツを処理および生成する。

【００１５】このシーケンスにおける処理の１つは、クライアントからのリクエストに含まれる搬送可能状態構造で受け取る状態情報を階層内の適切な１つのまたは複数のオブジェクトに非直列化するロード処理を含む。このロード処理におけるトラバース処理は、階層内に適切な制御オブジェクトを見つけるために、搬送可能状態構造の階層情報に従い制御オブジェクト階層をトラバースする。本発明のある実施形態において、受け取った状態情報は、初期状態から変化した制御オブジェクトの状態のみを含む（すなわち、この状態情報は、差分データのみを含む）。別の実施形態では、変化していようといまいと階層内のすべての制御オブジェクトのすべての状態、または１つ以上のサーバ側制御オブジェクトの状態情報および階層情報という何か別の組み合わせを含むことがある。

【００１６】上記シーケンスの別の処理は、状態が変化した制御オブジェクトからの状態情報を直列化する保存処理を含む。状態情報は、クライアントへ送信する搬送可能状態構造に状態情報を追加する。クライアントは、ウェブサーバへの次のリクエストでウェブサーバへその搬送可能構造を戻す。搬送可能状態構造を受信すると制御オブジェクト階層が再作成され、ロード処理により階層状態が以前のレスポンスの階層状態に戻る。

【００１７】別の実施形態において、搬送可能状態構造をクライアントに送信しないようにすることもできる。搬送可能状態構造は送信されずにサーバに残り、別の資源場所（例えば、別のサーバ）に転送されてもよい。例えば、ロードバランシングの場合、第１のサーバが、第１のHTTPトランザクションでブラウザと通信するが、第２のサーバが、第２のHTTPトランザクションでブラウザと通信することができる。したがって、搬送可能状態構造は、第１のサーバから第２のHTTPトランザクション処理する第２のサーバへと送られ、それによって、クライアントをバイパスすることができる。

【００１８】図１は、本発明のある実施形態におけるクライアントに表示するウェブページコンテンツを動的に生成するウェブサーバを示す。クライアント１００は、クライアント１００の表示装置上におけるウェブページ１０４を表示するブラウザ１０２を実行する。クライアント１００は、ビデオモニタのような表示装置を有するクライアントコンピュータシステムを含んでもよい。マイクロソフト社によって販売されている"INTERNET EXPLORER"ブラウザは、本発明のある実施形態におけるブラウザ１０２の一例である。他のブラウザの例として、"NETSCAPE NAVIGATOR" および "MOSAIC"などがあるが、これに限らない。例示したウェブページ１０４には、テキストボックス制御１０６および２つのボタン制御１０８、１１０が組み込まれている。ブラウザ１０２は、HTTPレスポンス１１２でウェブサーバ１１６からHTMLコードを受け取ることができ、HTMLコードにより記述されたウェブページを表示する。ある実施形態を参照してHTMLについて説明するが、特に制限されるものではなく、SGML（Standard Generalized Markup Language）、XML（eXtensible Markup Language）および、XMLベースのマークアップ言語であって、ポケットベル（登録商標）および携帯電話などの狭帯域無線装置の内容およびユーザインタフェースを特定するように設計されたWML（Wireless Markup Language）を含む他のオーサリング言語も本発明の範囲内であると考えられている。さらに、標準HTML３．２が、本明細書中に主に開示されているが、HTMLのいかなるバージョンも本発明の範囲内に含まれ得る。

【００１９】クライアント１００とウェブサーバ１１６との通信は、HTTPリクエスト１１４およびHTTPレスポンス１１２の一連の処理を用いて行うことができる。ある実施形態を参照してHTTPを説明するが、特に制限されるものではなく、S－HTTPを含めた他のトランスポートプロトコルも本発明の範囲内であると考えられている。ウェブサーバ１１６において、HTTPパイプラインモジュール１１８が、HTTPリクエスト１１４を受信し、URLを解析し、リクエストを処理する適切なハンドラを呼び出す。本発明のある実施形態において、異なるタイプの資源を扱う複数のハンドラ１２０がウェブサーバ１１６に備わっている。

【００２０】例えば、URLがHTMLファイルのような静的なコンテンツファイル１２２を特定する場合、ハンドラ１２０は、静的コンテンツファイル１２２にアクセスし、HTTPパイプライン１１８を介して静的コンテンツファイル１２２をHTTPレスポンス１１２でクライアント１００へ送る。あるいは、本発明のある実施形態において、URLがASP+リソースのような動的コンテンツファイル１２４を特定する場合、ハンドラ１２０は、動的コンテンツファイル１２４にアクセスし、動的コンテンツファイル１２４のコンテンツを処理し、ウェブページ１０４用の結果HTMLコードを生成する。本発明のある実施形態において、結果HTMLコードは、標準HTML３．２コードを含む。一般に、動的コンテンツファイルは、クライアントに表示すべきウェブページを記述するオーサリング言語を動的に生成するのに用いることができるサーバ側宣言データ格納部（例えば、ASP+リソース）である。そして、ウェブページ用のHTMLコードは、HTTPパイプライン１１８を通過し、HTTPレスポンス１１２でクライアント１００へ送られる。

【００２１】この処理の間、ハンドラ１２０はまた、開発労力を単純にするために予め開発された、または第３者コードのライブラリにアクセスすることができる。このようなライブラリの１つがサーバ側クラス制御ライブラリ１２６であり、ここから、ハンドラ１２０は、ユーザインタフェース要素を処理しウェブページに表示する結果HTMLデータを生成するサーバ側制御オブジェクトを例示することができる。本発明のある実施形態において、１つ以上のサーバ側制御オブジェクトは、動的コンテンツファイル１２４に記述されたウェブページ上に、可視的にまたは隠して、１つ以上のユーザインタフェース要素にマッピングする。

【００２２】これに対し、第２のライブラリは、マイクロソフト社からの"ACTIVEX"コンポーネントを含むライブラリのようなクライアント側制御クラスライブラリ１２８である。"ACTIVEX"制御は、クライアントおよび他のコンポーネントとの対話の仕方において一定の標準に従うCOM（コンポーネントオブジェクトモデル）オブジェクトである。クライアント側"ACTIVEX"制御は、例えば、クライアントに自動的にダウンロードされ、クライアントのウェブブラウザによって実行処理され得るCOMベースのコンポーネントである。サーバ側ACTIVEXコンポーネント（図示せず）は、株価検索アプリケーションまたはデータベースコンポーネントのサーバ側機能性を提供するような多様なサーバ側機能を果たすためにサーバ上で実行され得るCOMベースのコンポーネントである。ACTIVEXについては、「ACTIVEXおよびOLEの理解」（デビッド・チャペル、マイクロソフトプレス、１９９６年）により詳細に記載されている。

【００２３】"ACTIVEX"制御とは対照的に、動的コンテンツ資源１２４に特定される本発明の実施形態のサーバ側制御オブジェクトは、クライアント上のウェブページに組み込まれるユーザインタフェース要素に論理的に対応する。サーバ側制御オブジェクトはまた、例えば、HTMLタグと所与のクライアント側"ACTIVEX"制御をリファレンスするロケータを含み得る有効なHTMLコードを生成することができる。ブラウザが、すでに格納システム内にクライアント側"ACTIVEX"制御用コードを有している場合は、クライアント上のウェブページ内で"ACTIVEX"制御を実行処理する。そうでなければ、ブラウザは、ロケータによって特定された資源から"ACTIVEX"制御用コードをダウンロードし、そして、クライアント上のウェブ

ページ内で”ACTIVEX”制御を実行処理する。本発明の実施形態におけるサーバ側制御オブジェクトはまた、サーバ上で株価検索アプリケーションを実行するのに用いられるサーバ側”ACTIVEX”制御に対しイベントを提起することができる。

【００２４】ハンドラ１２０はまた、ウェブサーバ１１６上または別のアクセス可能ウェブサーバ上で実行処理する１つ以上の非ユーザインタフェースサーバコンポーネント１３０にアクセスする。株価検索アプリケーションまたはデータベースコンポーネントのような非ユーザインタフェースサーバコンポーネント１３０は、ハンドラ１２０によって処理される動的コンテンツファイル１２４において参照され、またはそれに関連付けられる。動的コンテンツファイル１２４で宣言された制御オブジェクトによって提起されたサーバ側イベントは、非ユーザインタフェースサーバコンポーネント１３０の適切なメソッドを呼び出すサーバ側コードによって処理され得る。その結果、サーバ側制御オブジェクトによって提供される処理は、ウェブページのユーザインタフェース要素の処理および生成をカプセル化することによって非ユーザインタフェースサーバコンポ－ネント１３０のプログラミングを簡単にし、これにより、非ユーザインタフェースサーバコンポーネント１３０の開発者は、ユーザインタフェース問題ではなく、アプリケーション固有の機能の開発に集中することができる。

【００２５】図２は、本発明のある実施形態におけるサーバ側制御オブジェクトを用いてのクライアント側ユーザインタフェース要素の処理および生成処理のフローチャートを示す。処理２００において、クライアントは、HTTPリクエストをサーバに送信する。HTTPリクエストは、ASP+リソースなどの資源を特定するURLを含む。処理２０２において、サーバは、HTTPリクエストを受信し、特定された資源を処理する適切なハンドラを呼び出す。HTTPリクエストは、１つ以上のサーバ側制御オブジェクトに関する状態情報および任意には階層情報を含んでいる搬送可能状態構造を含む。ただし、所与のページへの最初のHTTPリクエストは、その所与のページのサーバにおいて状態が変化していないことから、典型的には搬送可能状態構造を含んでいない。さらに、搬送可能状態構造は、プロパティデータタイプ情報および状態情報の有効性を確認する（すなわち、状態情報がクライアントで変造されなかったことを確認する）際にサーバを補助するためのインテグリティコードを含んでもよい。処理２０４は、特定された動的コンテンツファイル（例えば、ASP+リソース）の内容に基づきサーバ側制御オブジェクト階層を生成する。ASP+リソースは処理２０３において読み出される。処理２０５において、搬送可能状態構造で受け取った状態情報を階層内の適切なサーバ側制御オブジェクトにロードし、制御オブジェクトを以前の状態に戻す。

【００２６】処理２０６において、制御オブジェクト階層のサーバ側制御オブジェクトは、ポストバックイベントハンドリング、ポストバックデータハンドリング、状態管理およびデータ結合のうちの１つ以上の処理を行う。ユーザインタフェース要素からのポストバックイベントおよびデータ（まとめて「ポストバック入力」）は、クライアントからサーバへと送られ処理される。例えば、ポストバックイベントは、特に制限されるものではなく、クライアント側ボタン要素からの「マウスクリック」またはサーバに送られるクライアント側テキストボックス要素からの「データ変化」イベントを含んでもよい。ポストバックデータは、特に制限されるものではなく、例えば、テキストボックス要素またはドロップダウンボックスから選ばれた項目のインデックスにユーザによって入力されたテキストを含んでもよい。

【００２７】処理２０９において、プロパティ値、関連タイプ情報および階層情報をクライアントへ送信する搬送可能状態構造に格納する。処理２０８において、階層内の各サーバ側制御オブジェクトが、クライアント側ユーザインタフェース要素のウェブページでの表示のためのHTMLコードのようなオーサリング言語データを生成（またはレンダリング）するために呼び出される。用語「レンダリング」は、ユーザインタフェース上にグラフィックスを表示する処理を意味することがあるが、本明細書において「レンダリング」は、表示およびクライアント側機能のためのブラウザのようなクライアントアプリケーションによって解釈され得るオーサリング言語データの生成処理をも意味している。レンダリング処理２０８はまた、搬送可能状態構造を表しているオーサリング言語データを生成する。処理２０６およびレンダリング処理２０８のより詳細な説明は図６との関連で行われている。ある実施形態において、個々の制御オブジェクトにおけるrender（）メソッドの呼び出しは、ツリートラバーサルシーケンスを用いて行われる。すなわち、ページオブジェクトのrender（）メソッドの呼び出しは、階層内の適切なサーバ側制御オブジェクトにわたる反復トラバースになる。適切な制御オブジェクトのrender（）メソッドを呼び出す別の方法として、イベントシグナリングまたはオブジェクト登録手法などの方法を用いてもよい。括弧は、データ値と比較するなどのメソッドを示す「render（）」レベルを指定する。

【００２８】本発明のある実施形態において、個々のサーバ側制御オブジェクトの実際の作成は、サーバ側制御オブジェクトが処理２０６または２０８においてアクセスされる（ポストバック入力のハンドリング、状態のロード、制御オブジェクトからHTMLコードのレンダリングなど）まで遅らせてもよい。サーバ側制御オブジェクトが所与のリクエストのためにアクセスされることがない場合、制御オブジェクトの作成を遅らせ、不要な制御オブジェクト作成処理を排除することによって、サーバ処

理が最適化される。

【００２９】処理２１０において、搬送可能状態構造を含むオーサリング言語データ（例えば、HTMLコード）をHTTPレスポンスでクライアントに送信する。処理２１４において、クライアントは、表示されるべきウェブページに関連する搬送可能状態構造を含むHTMLコードを受信する。搬送可能状態構造は、次のHTTPリクエストでサーバに戻すためにクライアントに保存され得る。処理２１６において、クライアントシステムは、HTTPレスポンスから受け取ったHTMLコードに応じて新しいページのユーザインタフェース要素を組み入れる（例えば、表示する）。しかし、ユーザインタフェース要素の組み入れは、音声または触覚出力を提供する、メモリへの読出しおよび書込み、スクリプト処理の制御などの非表示処理を含んでもよいことを理解すべきである。処理２１２において、サーバ側制御オブジェクト階層を終了する。本発明のある実施形態において、階層内サーバ側制御オブジェクトは、関連付けられたASP+リソースを参照するHTTPリクエストに応答して作成され、オーサリング言語データ（例えば、HTMLデータ）のレンダリングが終わると破壊される。別の実施形態において、処理２１２は、処理２０８の後、処理２１０の前に行ってもよい。

【００３０】図３は、本発明のある実施形態において用いるウェブサーバでのモジュールの一例を示す。ウェブサーバ３００は、HTTPパイプライン３０４にHTTPリクエスト３０２を受け入れる。HTTPパイプライン３０４は、ウェブページ統計のロギング、ユーザ照合、ユーザアクセス権およびウェブページの出力キャッシュ化用モジュールなどの種々のモジュールを含んでもよい。ウェブサーバ３００によって受信される各入力HTTPリクエスト３０２は、最終的には、IHTTPハンドラクラス（ハンドラ３０６として図示）の特定のインスタンスによって処理される。IHTTPという接頭語は、そのクラスがHTTPハンドラのインタフェースであることを示している。ハンドラ３０６は、URLリクエストを解析し適切なハンドラファクトリ（例えば、ページファクトリモジュール３０８）を呼び出す。

【００３１】図３において、ASP+リソース３１０に関連付けられたページファクトリ３０８が呼び出され、ASP+リソース３１０において宣言されたオブジェクトの例示および構成をハンドリングする。ある実施形態において、ASP+リソースは、ファイルに特定の接尾語（または".aspx"のようなファイル拡張部分）を指定することによって認識または参照され得る。所与の".aspx"資源に対するリクエストがまず、ページファクトリモジュール３０８によって受信されると、ページファクトリモジュール３０８は、ファイルシステムを検索して適切なファイル（例えば、.aspxファイル３１０）を得る。このファイルは、リクエストを処理するサーバによって後に解釈またはアクセスされ得るテキスト（例えば、オーサリング言語データ）または別のフォーマットでのデータ（例えば、バイトコードデータまたはコード化されたデータ）を含んでもよい。物理的なファイルが存在する場合、ページファクトリモジュール３０８は、ファイルを開き、そのファイルをメモリに読み出す。ファイルが見つからない場合、ページファクトリモジュール３０８は、適切な「ファイルが見つからない」というエラーメッセージを戻す。

【００３２】ASP+リソース３１０をメモリに読み出した後、ページファクトリモジュール３０８は、ファイル内容を処理し、ページのデータモデル（例えば、スクリプトブロックのリスト、ディレクティブ、静的テキスト領域、階層サーバ側制御オブジェクト、サーバ側制御プロパティなど）を構築する。データモデルは、ページベースのクラスを拡張させるCOM+（Component Object Model+）クラスのような新たなオブジェクトクラスのソースリストを生成するのに用いられる。ページベースクラスは、基本ページオブジェクトの構造、プロパティおよび機能を規定するコードを含んでいる。次いで、ソースリストは、中間言語に動的にコンパイルされる。中間言語は、COM+ IL コード、Java バイトコード、Modula 3 コード、SmallTalk コードおよびVisual Basicコードなどの汎用またはカスタム指向言語コードを含んでもよい。別の実施形態において、中間言語処理を省き、固有の命令をソースリストまたはソースファイル（例えば、ASP+リソース３１０）から直接生成してもよい。制御クラスライブラリ３１２は、制御オブジェクト階層の生成に用いられる予め定義されたサーバ側制御クラスを得るためにページファクトリモジュール３０８によってアクセスされ得る。

【００３３】ページファクトリモジュール３０８は、図１のウェブページ１０４に相当するサーバ側制御オブジェクトであるページオブジェクト３１４を出力する。ページオブジェクト３１４およびその子オブジェクト（例えば、テキストボックスオブジェクト３１８、ボタンオブジェクト３２０および別のボタンオブジェクト３２２）が、制御オブジェクト階層３１６の一例である。他の制御オブジェクトの例は、本発明に従い考えることができ、特に制限されるものではなく、表１のHTML制御に対応するオブジェクトおよびカスタム制御オブジェクトを含む。ページオブジェクト３１４は、図１のウェブページ１０４に論理的に対応する。テキストボックスオブジェクト３１８は、図１のテキストボックス１０６に対応する。同様に、ボタンオブジェクト３２０は、図１の追加ボタン１０８に対応し、ボタンオブジェクト３２２は、図１の削除ボタン１１０に対応する。ページオブジェクト３１４は、サーバ上の他の制御オブジェクトと階層的に関連している。ある実施形態において、ページオブジェクトは、その子制御オブジェクトを階層的に含んでいるコンテナオブジェクトである。別の実施形態で

は、依存関係などの他の形態の階層関係を用いることができる。多数レベルの子オブジェクトを有するより複雑な制御オブジェクト階層において、１つの子オブジェクトが、他の子オブジェクトのコンテナオブジェクトであってもよい。

【００３４】上記の実施形態において、制御オブジェクト階層３１６の制御オブジェクトは、サーバ３００上で作成および実行され、各サーバ側制御オブジェクトは、クライアント上の対応するユーザインタフェース要素を「ミラーする」。本実施形態のサーバ側制御オブジェクトはまた、協働して、HTTPリクエスト３０２からの入力をハンドリングし、サーバ側制御オブジェクトの状態を管理し、サーバ側データベースとのデータ結合を行い、クライアントでの結果ウェブページの表示に用いるオーサリング言語データ（例えば、HTMLコード）を生成する。結果オーサリング言語データは、サーバ側制御オブジェクト階層３１６から生成（すなわち、レンダリング）され、HTTPレスポンス３２４でクライアントに送信される。例えば、結果HTMLコードは、いかなる有効なHTML構成も具現化することができ、ACTIVEXタイプの制御、ＪＡＶＡ（登録商標）アプレット、スクリプトおよびその他、ブラウザによって処理されたとき、クライアント側ユーザインタフェース要素（例えば、制御ボタン、テキストボックスなど）を生み出すいかなるウェブ資源も参照することができる。

【００３５】ASP+リソース３１０でなされた宣言によって、サーバ側制御オブジェクトは、非ユーザインタフェースサーバコンポーネント３３０とクライアント側ユーザインタフェース要素との対話のために、１つ以上の非ユーザインタフェースサーバコンポーネント３３０にアクセスすることができる。例えば、ポストバック入力に応答して、サーバ側制御オブジェクトは、サーバ側イベントをそれらのイベント用に登録された非ユーザインタフェースサーバコンポーネントに対し提起することができる。このように、非ユーザインタフェースサーバコンポーネント３３０は、ユーザとの対話を、ユーザインタフェース要素を介して、これらの要素を表示および処理するのに必要なコードをプログラミングすることなく行うことができる。

【００３６】要するに、本発明の実施形態は、クライアントに送るHTMLコードを生成するためにサーバ上で作成および実行処理されるサーバ側制御オブジェクトを含む。HTMLコードは、例えば、いかなる有効なHTML構成も具現化することができ、ACTIVEXタイプの制御、JAVA アプレット、スクリプトならびにクライアントでのユーザインタフェースボタンおよび他のユーザインタフェース要素を生成する他のいかなるウェブ資源も参照することができる。クライアントでのユーザは、サーバ側制御オブジェクトに論理的に対応するこれらのユーザインタフェース要素と対話し、リクエストをサーバに送り返すことができる。サーバ側制御オブジェクトは、サーバ上で再作成され、クライアントにレスポンスとして送信すべき次のラウンドのHTMLコードを生成するように、ユーザインタフェース要素のデータ、イベントおよび他の特性を処理する。

【００３７】図４を参照すると、本発明の実施形態のコンピュータシステムの一例は、プロセッサユニット４０２、システムメモリ４０４およびシステムメモリ４０４を含む種々のシステムコンポーネントをプロセッサユニット４００に接続するシステムバス４０６を含んでいる従来のコンピュータシステム４００という形態の汎用コンピュータ装置を含んでいる。システムバス４０６は、メモリバスまたはメモリコントローラ、種々のバスアーキテクチャを用いるペリフェラルバスおよびローカルバスを含む幾つかのタイプのバス構造のいずれであってもよい。システムメモリは、再生専用メモリ（ROM）４０８およびランダムアクセスメモリ（RAM）４１０を含んでいる。コンピュータシステム４００内の要素間での情報の転送を助ける基本ルーチンを含んでいる基本入力／出力システム４１２（BIOS）は、ROM４０８に格納されている。

【００３８】コンピュータシステム４００は、さらに、ハードディスクの読み出しおよび書き込みを行うハードディスクドライブ４１２、着脱可能な磁気ディスク４１６の読出しおよび書込みを行う磁気ディスクドライブ４１４およびCD ROM、DVDまたは他の光学媒体のような着脱可能な光ディスク４１９の読出しおよび書込みを行う光ディスクドライブ４１８を含んでいる。ハードディスクドライブ４１２、磁気ディスクドライブ４１４および光学ディスクドライブ４１８は、それぞれ、ハードディスクドライブインタフェース４２０、磁気ディスクドライブインタフェース４２２および光ディスクドライブインタフェース４２４によってシステムバス４０６接続されている。ドライブおよびその関連コンピュータ読み取り可能媒体が、コンピュータシステム４００のコンピュータ読み取り可能命令、データ構造、プログラムおよび他のデータの不揮発性記憶部を提供している。

【００３９】本明細書に記載の上記環境例では、ハードディスク、着脱可能な磁気ディスク４１６および着脱可能な光ディスク４１９を用いているが、データ保存可能な他のタイプのコンピュータ読み取り可能媒体を上記システム例に用いることができる。上記動作環境例に用いることができるこれらの他のタイプのコンピュータ読み取り可能媒体は、例えば、磁気カセット、フラッシュメモリカード、デジタルビデオディスク、ベルヌイ(Bernoulli)カートリッジ、ランダムアクセスメモリ（RAM）および再生専用メモリ（ROM）などがある。

【００４０】多数のプログラムモジュールが、ハードディスク、磁気ディスク４１６、光ディスク４１９、ROM４０８またはRAM４１０に格納され得、これらは、オペ

レーティングシステム４２６、１つ以上のアプリケーションプログラム４２８、他のプログラムモジュール４３０およびプログラムデータ４３２を含む。ユーザは、コマンドおよび情報をコンピュータシステム４００にキーボード４３４およびマウス４３６または他のポインティング装置などの入力装置によって入力することができる。他の入力装置としては、例えば、マイクロフォン、ジョイステック、ゲームパッド、サテライトディッシュおよびスキャナなどがある。これらおよび他の入力装置は、システムバス４０６に接続されているシリアルポートインタフェース４４０を介して処理装置４０２に接続されていることが多い。しかし、これらの入力装置はまた、パラレルポート、ゲームポートまたはユニバーサルシリアルバス（USB）などの他のインタフェースによって接続されていてもよい。モニタ４４２または他のタイプの表示装置もまた、ビデオアダプタ４４４などのインタフェースを介してシステムバス４０６と接続している。モニタ４４２に加えて、コンピュータシステムは、典型的には、スピーカおよびプリンタなどの他の周辺出力装置（図示せず）を含む。

【００４１】コンピュータシステム４００は、リモートコンピュータ４４６のような１つ以上のリモートコンピュータへの論理接続を用いたネットワーク化された環境で動作することができる。リモートコンピュータ４４６は、コンピュータシステム、サーバ、ルータ、ネットワークPC、ピア（peer）装置、または他の共通ネットワークノードであり得、典型的にコンピュータシステム４００との関連で上述した要素の多くまたはすべてを含む。ネットワーク接続は、ローカルエリアネットワーク（LAN）４４８およびワイドエリアネットワーク（WAN）４５０を含む。このようなネットワーク環境は、オフィス、企業規模コンピュータネットワーク、イントラネットおよびインターネットにおいて珍しいものではない。

【００４２】LANネットワーク環境で用いるとき、コンピュータシステム４００は、ネットワークインタフェースまたはアダプタ４５２を介してローカルネットワーク４４８に接続される。WANネットワーク環境で用いるとき、コンピュータシステム４００は、典型的に、インターネットのようなワイドエリアネットワーク４５０による通信を確立するためのモデム４５４または他の手段を含む。モデム４５４は内蔵または外付けのいずれでもよく、シリアルポートインタフェース４４０を介してシステムバス４０６と接続されている。ネットワーク化された環境において、コンピュータシステム４００に関連して述べたプログラムモジュールまたはその一部は、リモートメモリ記憶装置に記憶されてもよい。図示されたネットワーク接続は例であって、コンピュータ間の通信リンク確立のために他の手段を用いることができる。

【００４３】本発明の実施形態において、コンピュータ４００は、ウェブサーバを表し、CPU４０２が、記憶媒体４１６、４１２、４１４、４１８、４１９またはメモリ４０４のうち少なくとも１つに記憶されたASP+リソース上でページファクトリモジュールを実行処理する。HTTPレスポンスおよびリクエストは、クライアントコンピュータ４４６に接続されたLAN４４８により通信される。

【００４４】図５は、本発明のある実施形態におけるページオブジェクトおよび他の制御オブジェクトのサーバ側処理を表すプロセスフローチャートである。処理５００において、ページオブジェクト構築部が、ページファクトリモジュール３０８によって呼び出される（図３参照）。その結果、ページオブジェクト（例えば、図３におけるページオブジェクト３１４を参照）は、クライアント上のウェブページユーザインタフェース要素を「ミラーする」ために作成される。処理５０２において、ページファクトリモジュールは、クライアントから受け取ったHTTPリクエストの段階的な処理を始めるページオブジェクトのProcessRequestメンバ関数を呼び出す。本発明の一実施形態の第１の段階において、サーバ側作成処理(図示せず)は、ページオブジェクトの制御オブジェクト階層に含まれる子孫サーバ側制御オブジェクトの作成であり得る。すなわち、子制御オブジェクトの構築部がHTTPリクエスト処理の処理生存期間の間制御オブジェクトを作成するために繰り返し呼び出される。

【００４５】しかし、別の実施形態において、子制御オブジェクトの作成は、制御オブジェクトが所与の処理ステップ（例えば、ポストバックイベントのハンドリング、ポストバックデータのハンドリング、ビューステートのローディングおよび保存、データ結合の解析または対応するユーザインタフェース要素のHTMLコードのレンダリング）に必要とされるまで遅らせることができる。「制御オブジェクト作成を遅らせる」後者の実施形態は、不必要なCPUおよびメモリの利用を減らすことができるので最適である。例えば、クライアントから受け取ったユーザ入力イベントが、全く異なるウェブページの作成ということになる場合がある。この場合、直ちに制御オブジェクト階層の終了させ、新たなページの新たな異なる制御オブジェクト階層を例示することになるイベントを処理するためだけに、以前のページの制御オブジェクト階層全体を例示する必要はない。

【００４６】ページオブジェクトのProcessRequestメソッドのサーバ呼び出しに応答して、処理５０４～５２０は、所与のHTTPリクエストのデータなどに応じ、データページオブジェクトおよび個々の子孫制御オブジェクトによって実行処理することができる。本発明のある実施形態において処理５０４～５２０を図５の順序で各個々のオブジェクトに対し行う。しかし、１つのオブジェクトに対する所与の処理は、HTTPリクエストによっては、別のオブジェクトの所与の処理に関して順番に行われなかったり、全く処理が行われないこともある。例えば、

第１のオブジェクトは、初期化処理５０４およびそのロード処理５０６を行い、ポストバックデータ処理５０８を開始し、その後、遅れた制御オブジェクト作成により子孫制御オブジェクトが、それ自体の初期化処理５０４およびロード処理５０６を行う。ページオブジェクトおよび子孫制御オブジェクトによる処理の順番は、これには限らないが、HTTPリクエストにおけるデータの性質、制御オブジェクト階層の構成、制御オブジェクトの現在の状態および制御オブジェクト作成が遅れて行われるかどうかを含む種々の要因による。

【００４７】初期化処理５０４は、動的コンテンツファイルにおける初期化に関する任意のサーバ側コードを実行処理することによって制御オブジェクトが作成された後、制御オブジェクトを初期化する。このようにして、各サーバ側制御オブジェクトは、動的コンテンツファイルで宣言された特定のサーバ側機能でカスタマイズされ得る。本発明の実施形態において、動的コンテンツコードは、サーバ上のASP+リソースでページ開発者によって宣言されたベースページ制御クラスをカスタマイズまたは拡張することが意図された。ASP+リソースがコンパイルされると、宣言されたコードは、適切な初期コード（例えば、ページオブジェクトおよび子孫制御オブジェクトの初期化（）メソッド）に含まれている。初期化処理５０４は、このコードを実行処理して、ページベースクラスおよび子孫制御オブジェクトのベースクラスをカスタマイズまたは拡張する。

【００４８】本発明のある実施形態において、サーバ側制御オブジェクトの状態管理は、サーバ側制御オブジェクトを以前の状態に戻すことによってクライアント・サーバシステムの無状態モデルを収納するために搬送可能状態構造を用いるロード処理５０６および保存処理５１６にサポートされる。ある実施形態では、状態は、一対のHTTPリクエスト／レスポンスの隠れた１つ以上のHTMLフィールドでサーバを往復するが、クッキーおよび可視フィールドなどの他の搬送可能状態構造も本発明の範囲内であると考えられる。

【００４９】クライアントとサーバ間の現在のページに関する所与のリクエストおよびレスポンスによる一連の処理において、１つ以上の制御オブジェクトの状態は、先のリクエストの処理後に保存処理５１６によって搬送可能状態構造に記録される。本発明のある実施形態において、階層情報またはサーバが適切な制御オブジェクトと所与の状態とを関連付けることを可能にする制御オブジェクト識別子を含む追加の状態情報もまた、搬送可能状態構造に含まれる。次のHTTPリクエストにおいて、状態情報は、搬送可能状態構造でサーバに戻される。サーバは、受信した搬送可能状態構造から状態情報を抽出し、状態データを制御オブジェクト階層内の適切な制御オブジェクトにロードし、各制御オブジェクトを先のHTTPレスポンス以前に存在したような状態に戻す。現在のリクエストに対する処理後、１つ以上のサーバ側制御オブジェクトの状態は、再度保存処理５１６によって搬送可能状態構造に記録され、次のHTTPレスポンスで搬送可能状態構造をクライアントに戻す。

【００５０】ロード処理５０６の結果、各サーバ側制御オブジェクトを先のHTTPリクエスト以前の状態と同じ状態にする。例えば、テキストボックス制御オブジェクトが、先のHTTPレスポンス以前に"JDoe"と同等のプロパティ値を含む場合、テキストストリング"JDoe"をそのプロパティ値にロードすることなどによって、ロード処理５０６は、同じ制御オブジェクトを先の状態に戻す。さらに、所与のオブジェクトの状態が記憶され回復されたかどうかについても構成可能である。

【００５１】本発明の一実施形態を要約すると、１つ以上のサーバ側制御オブジェクトの状態が処理後「保存」される。保存された状態情報は、レスポンスによりクライアントに送信される。クライアントは、次のレスポンスで保存された状態情報をサーバ側に戻す。サーバは、階層の状態が以前の状態に戻るように、新たに例示されたサーバ側制御オブジェクト階層に状態情報をロードする。

【００５２】別の実施形態では、サーバ上、またはサーバからクライアント、そしてサーバへ戻るという往復の間、サーバによってアクセス可能な幾つかの他のウェブロケーションに状態情報を保持できる。クライアントリクエストをサーバが受信した後、この状態情報は、サーバによって取り出され、制御オブジェクト階層内の適切なサーバ側制御オブジェクトにロードされ得る。

【００５３】処理５０８において、HTTPリクエストから受け取ったポストバックデータが処理される。ポストバックデータは、対になったキー値、階層表示（例えば、XML）またはRDF（Resource Description Framework）のような他のデータ表示でのHTTPリクエストのペイロードに含まれ得る。処理５０８は、ペイロードを構文解析してサーバ側制御オブジェクトの一意の識別子を識別する。識別子（例えば、"page1:text1"）を見つけ、識別されたサーバ側制御オブジェクトが制御オブジェクト階層に存在する場合、対応するポストバックデータがその制御オブジェクトへと渡される。例えば、図１を参照すると、テキストボックス１０６およびテキスト"JDoe"に関連する一意の識別子は、HTTPリクエスト１１４のペイロードでウェブサーバ１１６へ送られる。処理５０８はHTTPリクエスト１１４のペイロードを構文解析し、テキストボックス１０６の一意の識別子とその関連値（すなわち"JDoe"）を得る。それから処理５０８は、テキストボックス１０６の一意の識別子を解析し、対応するサーバ側制御オブジェクトを識別し、処理するオブジェクトに"JDoe"値を渡す。

【００５４】ロード処理５０６に関して説明したように、サーバ側制御オブジェクトのプロパティ値は、以前

の状態に戻され得る。ポストバックデータを受け取ると、サーバ側制御オブジェクトは、渡されたポストバック値が対応する先のプロパティ値を変化させたかどうかを判断する。変化した場合には、関連制御オブジェクトのデータ変化を示す変化リストにその変化をロギングする。すべてのポストバックデータが制御オブジェクト階層内で処理された後、制御オブジェクトメソッドが呼び出され、サーバ上で起動している株価検索アプリケーションのような１つ以上の非ユーザインタフェースサーバコンポーネントに対し、１つ以上のポストデータ変化イベントを提起し得る。ポストデータ変化イベントは、例えば、ポストバックデータがサーバ側制御オブジェクトのプロパティを変化させたということを示しているイベントである。例示的な実施形態において、このようなイベントは、システム提供イベント待ち行列に送られ、イベントを処理するよう登録されたサーバコンポーネントを呼び出すことができる。このように、サーバ側非ユーザインタフェースサーバコンポーネントは、サーバ側制御オブジェクトのデータの変化によってトリガされたイベントに応答することができる。アプリケーション提供イベント待ち行列、ポーリング、および処理割込みを用いることを含む、イベントを実行する別の方法もまた本発明の範囲内で考えることができる。

【００５５】処理５１０において、ポストバックイベントをハンドリングする。ポストバックイベントは、HTTPリクエストのペイロードで通信される。処理５１０は、そのイベントが向けられているサーバ側制御オブジェクトを識別する特定のイベントターゲット（例えば、本発明のある実施形態では、"##EVENTTARGET"とラベル付けされている）を構文解析する。さらに、処理５１０は、探し当てたイベント引数がある場合はそれを構文解析し、イベント引数（例えば、本発明のある実施形態では、"##EVENTARGUMENT"とラベル付けされている）を特定されたサーバ側制御オブジェクトに与える。制御オブジェクトは、動的コンテンツファイルに関連する非ユーザインタフェースサーバコンポーネント（例えば、サーバ側株価検索アプリケーション）によって処理するイベントを立ち上げる。

【００５６】処理５１２は、サーバ側制御オブジェクトとサーバがアクセス可能な１つ以上のデータベースとの間のデータ結合を解明する。本発明のある実施形態では、サーバ側制御オブジェクトのプロパティは、サーバ側アプリケーションデータベースの表のような親データ結合コンテナのプロパティと関連付けられ（またはデータ結合され）得る。データ結合処理６１２の間に、ページフレームワークは、対応する親データ結合コンテナプロパティの値を有するデータ結合された制御オブジェクトプロパティを更新することができる。このように、次のレスポンスにおけるウェブページ上のユーザインタフェース要素は、更新されたプロパティ値を正確に反映する。なぜなら、ユーザインタフェース要素が対応する制御オブジェクトプロパティは、データ結合処理５１２の間に、自動的に更新されたからである。同様に、制御オブジェクトプロパティはまた、親データ結合コンテナフィールドに更新され得、これにより、サーバ側制御オブジェクトからのポストバック入力でサーバ側アプリケーションデータベースを更新する。

【００５７】処理５１４は、制御オブジェクト状態が保存され出力がレンダリングされる以前に実行処理され得る多数の更新処理を行う。処理５１６は、制御オブジェクト階層内の１つ以上の制御オブジェクトから状態情報（すなわち、ビューステート）を要求し、状態情報を格納し、HTTPレスポンスペイロードでクライアントへ送られる搬送可能状態構造に挿入する。例えば、「グリッド」制御オブジェクトは、値のリストの現在のインデックスページを保存し、「グリッド」制御オブジェクトは、次のHTTPリクエスト（すなわち、処理５０６）の後この状態に戻ることができる。上記のように、ビューステート情報は、クライアントによる次のアクション以前（例えば、HTTPレスポンスがクライアントに送られる以前）の制御オブジェクト階層の状態を表す。ビューステート情報が戻ってくると、それは、制御オブジェクト階層を、クライアントポストバック入力処理またはデータ結合以前の先の状態にするのに用いられる。

【００５８】レンダリング処理５１８は、HTTPレスポンスでクライアントへ送る適切なオーサリング言語出力（例えば、HTMLデータ）を生成する。レンダリングは、すべてのサーバ側制御オブジェクトのトップダウン階層ツリーウォークおよび埋め込まれたレンダリングコードにより行われる。処理５２０は、任意の最終のクリーンアップ作業（例えば、ファイルを閉じたりデータベースの接続）を行い、制御オブジェクト階層を終了させる。次いで、処理は５０２に戻り、処理５２２を行い、そこでページオブジェクトは、その破壊部を呼び出すことによって終了する。

【００５９】図６は、本発明のある実施形態における動的コンテンツファイル（例えば、ASP+リソース）の部分の一例を示す。ASP+リソース６００のライン１は、HTMLファイルの開始タグであり、ASP+リソース６００においてリテラルとして宣言される。リテラルは、サーバ側制御オブジェクト階層内のサーバ側リテラル制御オブジェクトに対応する。ライン１に対応するリテラル制御オブジェクトには、ASP+リソース６００での宣言による第１の制御オブジェクトであることから、インデックス「０」を与える。レンダリング時間において、リテラル制御オブジェクトは、単に"<html>"テキストおよびHTTPレスポンスに含むための新たなラインを生成するだけである。ASP+リソース６００のライン２から１２は、コード宣言ブロックを表し、これは、サーバ上で実行処理される（すなわち、ライン２の"runat =server"属性で示

されたように）。本ASP+リソース６００の例において、コード宣言ブロックの結果、制御オブジェクト階層内のサーバ側制御オブジェクトの例示は起こらない。そうではなくて、コード宣言ブロックにより、サーバ側コードが、ライン１５で宣言されたサーバ側制御オブジェクトと「ワイヤで繋がれ」たり、または関連付けられたりする。ASP+リソース６００のライン１３は、HTMLファイルの本文の開始タグであり、リテラルとしてASP+リソース６００で宣言され、インデックス「１」を有するリテラル制御オブジェクトとなる。

【００６０】ライン１４での宣言は、制御オブジェクト階層に例示されるサーバ側フォーム制御オブジェクト（インデックス＝「２」）を宣言する。ライン１７および１８でのタグは、制御オブジェクト階層内でのサーバ側リテラル制御オブジェクト（インデックス＝「３」）と表される終了タグである。ライン１６の終了タグは、フォーム制御オブジェクトに対応する<form>宣言を終了する。

【００６１】次の階層レベルで、ライン１５の宣言は、HTML制御タグ"span"を用い、"Message"という識別子属性("id")を有するサーバ側ラベル制御オブジェクト（インデックス＝「１」）を宣言する。この階層レベルのインデックス「０」および「２」はそれぞれ、先行のホワイトスペースリテラル（例えば、タブ、新たなライン、スペースなど）のリテラル制御オブジェクト（インデックス＝「０」）および次のホワイトスペースリテラルのリテラル制御オブジェクト（インデックス＝「２」）に割り当てられる。

【００６２】図７は、図６の動的コンテンツファイルの部分例に応答して１つ以上のサーバ側制御オブジェクトによって生成される結果コードを示す。本説明を簡単にするために、フォーム、状態管理およびスパンセクションのみをHTMLコード部７００に示している。<html>および<body>のような他のタグは、HTMLコード部７００に含まれていないが、これらのタグのHTMLコードは、図６のASP+リソースからの１組の完全なHTMLコードに含まれ得る。

【００６３】ライン１および５は、HTMLコードのフォームセクションの開始および終了タグであり、図６のライン１４から１６で宣言されたサーバ側フォーム制御オブジェクトによって生成される。ライン４は、図６のライン１５で宣言された最終ポストバック処理の日付を表示しているラベル（すなわち、"Last Post Back: 5/6/99"）のHTMLコードを含んでいる。

【００６４】ライン２および３の隠されたフィールドは、本発明による搬送可能状態構造の一実施形態を表している。ライン２において、ASP+リソース６００で宣言されたサーバ側制御オブジェクトの状態情報は、"##VIEWSTATE"という名の隠れたフィールドに記録される。"##VIEWSTATE"フィールド値は、状態値、プロパティタイプおよび制御オブジェクト階層内の制御オブジェクトの階層情報を表すテキストストリングである。階層情報はまた、ロード処理が、階層をトラバースし、所与の状態値を階層内の適切な制御オブジェクトのプロパティにロードするために"##VIEWSTATE"フィールドに含まれている。

【００６５】本発明のある実施形態では、クライアントはHTTPリクエスト／レスポンス対シーケンスを用いてウェブサーバと対話する。クライアントへのレスポンスとサーバが受信する次のリクエストとの間に、サーバは、所与のASP＋リソースまたは所与のクライアントとの接続に関連するサーバ側制御オブジェクトの状態を維持しなくてもよい。維持する代わりに、所与の制御オブジェクト階層の状態情報を搬送可能状態構造で（例えば、HTTPレスポンスで）クライアントに送り、搬送可能状態構造で（例えば、次のHTTPリクエストで）サーバに戻される。

【００６６】ライン３は、"434333433"という値を有する"##VIEWSTATE"という名の隠れたファイルを表す。"##VIEWSTATEMAC"は、##VIEWSTATE値がクライアントで変造されていないことを確認するためにロード処理によって用いられる搬送可能状態構造におけるインテグリティコードである。##VIEWSTATEMAC値は、最初、##VIEWSTATE値の内容からサーバで計算され、レスポンスによりクライアントに送られる。クライアントが搬送可能状態構造を戻すとき、ウェブサーバは、受信した##VIEWSTATE値の内容から新たなインテグリティコードを計算する。受信したインテグリティコードおよび新たに計算されたコードが等しければ、ウェブサーバは、受信した##VIEWSTATE値は有効または変造されていない（すなわち、クライアントに先に送った##VIEWSTATE値と同じ）と判断する。本発明のある実施形態では、MD5アルゴリズムを用いて、インテグリティコードを計算する。MD５は、デジタル署名の作成に使用するためにドナルド・リベスト教授によって１９９１年に作成されたアルゴリズムである。MD5は、一方向のハッシュ機能、すなわち、メッセージを受け取り、それをデジタルの固定ストリングに変換するもので、メッセージダイジェストとも呼ばれる。このようなメソッドは、アクセス権のない個人またはプログラムによる不正操作を減らす秘密の鍵の使用を含んでもよい。しかし、他の実施形態では、MD4アルゴリズムまたは他の任意のハッシュアルゴリズムに基づくコードを生成すること、あるいは##VIEWSTATEフィールドの長さを含む別のインテグリティコード化技術を含んでもよい。

【００６７】本発明のある実施形態では、ライン２の##VIEWSTATEフィールドの構造は、制御オブジェクトおよびそれらの状態（例えば、状態値およびそれに関連するプロパティデータタイプ）の階層的入れ子状態を表す。##VIEWSTATEフィールドの一例に用いるタグを（表１）

に示す。"/"を含んでいるタグは終了タグを表す。　　【表１】

【００６８】

タグ	説明
<s>, </s>	ストリング値
<a,x>, 	x 要素を有するアレイ
<i>	整数値
<h,x>, </H>	x テーブルエントリを有するハッシュテーブル
<K>, </K>	ハッシュテーブルキー
<V>, </V>	ハッシュテーブル値
,	ブーリアン値
<d>, </d>	日付／時間値
<c>, </c>	カレンシ値
<A>, </A>	アレイリスト
<n>, </n>	ヌル値

【００６９】本明細書中ではコード化実施形態の一例を示すが、別の実施形態では、他のコード化方法を用いることができることを理解すべきである。例えば、階層情報、状態値およびそれらの関連プロパティデータタイプを各サーバ側制御オブジェクトの一意の階層識別子を用いて指定することもできるし、あるいは、XML関連データフォーマットを用いて、搬送可能状態構造のデータを表すこともできる。さらに、公知の暗号化および圧縮技術を用いて、安全性を確保し、搬送可能状態構造のサイズを減らすこともできる。

【００７０】図８は、本発明のある実施形態における図６のASP＋リソースに対応する制御オブジェクト階層および図７の##VIEWSTATEフィールド値を示す。制御オブジェクト階層の制御オブジェクトコンポーネントではないが、ファイルチェックブロック８０２を図７に示すような##VIEWSTATEフィールド構造に基づく階層の最上レベルとして示している。ファイルチェックブロック８０２のより詳細な説明は、図９に関して行われている。あるいは、ファイルチェックブロック８０２を省いたり、ページ制御オブジェクト８０４に対応する階層レベルと一緒にすることも本発明の範囲内であると考えられる。

【００７１】ページ制御オブジェクト８０４が、制御オブジェクト階層の最上レベルとして例示され、結果のウェブページ自体に対応する特別制御オブジェクトに対応する。本発明のある実施形態では、ページオブジェクト８０４を例示させるための宣言（すなわち、"runat = server"属性を有する宣言）は特に必要とされない。

【００７２】次の階層レベルにおいて、制御オブジェクト８０６、８０８、８１０および８１２が、ページオブジェクト８０４に含まれる、または階層的に関連のある子制御オブジェクトとして例示されている。１つの階層レベルの各制御オブジェクトはASP＋リソースにおける上から下への順番に基づきゼロベースのインデックスを与えられている。例えば、リテラル制御オブジェクト階層８０６は、図６のASP＋リソース６００のライン１における"<html>"テキストおよび付随のホワイトスペースに対応している。したがって、リテラル制御オブジェクト８０６は、「０」というインデックスを与えられている。本発明の上記実施形態では、ライン２～１２から拡張されるコード宣言ブロックは、対応する制御オブジェクトの例示にはならない。リテラル制御オブジェクト８０８は、図６のASP＋リソース６００のライン１３における"<body>"テキストおよび付随のホワイトスペースに対応している。したがって、リテラル制御オブジェクト８０８は、「１」というインデックスを与えられている。フォーム制御オブジェクト８１０は、図６のASP＋リソース６００のライン１４～１６におけるフォーム宣言に対応している。したがって、フォーム制御オブジェクト８１０は「２」というインデックスを与えられている。リテラル制御オブジェクト８１２は、図６のASP＋リソース６００のライン１７および１８における終了タグおよび付随のホワイトスペースに対応している。したがって、リテラル制御オブジェクト８１２は、「３」というインデックスを与えられている。

【００７３】次の階層レベルにおいて、制御オブジェクト８１４、８１６および８１８が、フォームオブジェクト８１０に含まれる、または階層的に関連のある子制御オブジェクトとして例示されている。リテラル制御オブジェクト階層８１４は、図６のASP＋リソース６００のライン１５におけるスパン宣言に先立つホワイトスペースに対応している。したがって、リテラル制御オブジェクト８１４は、ASP＋リソースにおけるこの階層レベルとの関連において遭遇する第１のリテラルテキスト（ホワイトスペースであるが）であるから、「０」というインデックスを与えられている。同様に、リテラル制御オブジェクト８１８は、図６のASP＋リソース６００のラ

イン１５におけるスパン宣言に続くホワイトスペースに対応している。したがって、リテラル制御オブジェクト８１８は、「２」というインデックスを与えられている。ラベル制御オブジェクト８１６は、図６のASP＋リソース６００のライン１５における宣言に対応している。したがって、ラベル制御オブジェクト８１６は「１」というインデックスを与えられている。

【００７４】図９は、図７の##VIEWSTATEフィールド値の入れ子版を示す。本明細書中において##VIEWSTATEフィールド値の階層的性質の説明を簡単にするために入れ子版を記載している。入れ子版９００のライン１は、図８の入れ子になったファイルチェックブロック８０２の始まりに対応する。ライン１は、２要素アレイを特定し、第１の要素としてライン２のストリングと第２の要素としてライン３の２要素アレイを含んでいる。ライン２はファイルチェックコードを表すストリング値を特定する。本発明のある実施形態では、ファイルチェックコードは、関連のASP+リソース（例えば、図６のASP+リソース）のハッシュ値として計算される。ライン３で始まるアレイは、次の階層レベルの状態情報の始まり、すなわち図８のページ制御オブジェクト８０４を表している。ライン４のゼロ値は、ページオブジェクトそれ自体のプロパティに関する状態情報を保持することができるライン３の２要素アレイの第１の要素を表している。上記実施形態において、ページオブジェクトの状態情報は記録されていないので、その値は、”null”（ゼロ）である。

【００７５】ライン５に示すように、ライン３の２要素アレイの第２の要素は、ページ制御オブジェクトの子供（すなわち、図８の制御オブジェクト８０６、８０８、８１０および８１２）の状態情報を含むハッシュテーブルを示している。ライン５のハッシュテーブルは、タグ<H1>に「１」と示す１つのハッシュテーブルエントリを含む。ライン６および８の<K>、</K>タグは、ハッシュテーブルエントリの鍵、すなわち、図８のフォーム制御オブジェクト８１０のインデックスに相当するライン７に示す整数値「２」を含む。図９に示す##VIEWSTATE値は、図８のリテラル制御オブジェクト８０６、８０８および８１２の状態情報を含んでいない。あるいは、状態情報を保存した他の制御オブジェクトまたは子制御オブジェクトが、フォーム制御オブジェクト８１０として同じ階層レベルに存在する場合は、このレベルのハッシュテーブルは、このレベルの追加の制御オブジェクトを収納するための１つ以上のハッシュテーブルエントリを有していてもよく、各追加ハッシュテーブルエントリへの鍵は、対応する制御オブジェクトのインデックスであり得る。

【００７６】ハッシュテーブルエントリ値は、ライン９および２４の<V>、</V>タグに含まれ、フォーム制御オブジェクトとその子オブジェクトの状態を規定する別の２要素アレイを含む。そのアレイの第１の要素は、フォーム制御オブジェクト（すなわち、本実施形態においてゼロ値で表されている）の保存された状態情報を特定する。

【００７７】第２の要素は、フォーム制御オブジェクトの子オブジェクトの状態情報を表す別の１エントリハッシュテーブルを特定する。ライン１３および１５の<K>、</K>タグは、１つのハッシュテーブルエントリの鍵、すなわち、図８のラベル制御オブジェクト８１６のインデックスに相当するライン１４に示す整数値「１」を含む。図９に示す##VIEWSTATE値は、図８のリテラル制御オブジェクト８１４および８１８の状態情報を含んでいない。

【００７８】ハッシュテーブルエントリ値は、ライン１６および２１の<V>、</V>タグに含まれ、ラベル制御オブジェクトとその子オブジェクトの状態を規定する別の２要素アレイを含む。そのアレイの第１の要素は、ストリング値が、ライン１８で特定された”InnerHtml=5/16/99”である、ラベル制御オブジェクトのプロパティの保存された状態情報を特定する。プロパティ名、”InnerHtml”は、状態値”5/16/99”と等しく、「ストリング」タイプである。セミコロンは、所与のプロパティの状態値の終了を表す。ライン１８のラベル制御オブジェクトが、状態が保存された追加のプロパティを含んでいる場合、各状態データ値およびその関連プロパティは、終了セミコロンによって境界が定められる。アレイの第２の要素と同じようにライン１９に示されたゼロ値は、ラベル制御オブジェクトが子オブジェクトを含んでいないことを示す。ライン２０～２７は図９の先行する開始タグの終了タグである。

【００７９】別の実施形態では、有限オブジェクト直列化(Limitted Object Serialization(LOS))フォーマットと呼ばれる直列化フォーマットを搬送可能状態構造に用いる。一般に、LOSフォーマットは、各ハッシュエントリが、制御オブジェクトのプロパティの状態情報か、子制御オブジェクトの入れ子のハッシュテーブルのいずれかを含んでいる各制御オブジェクトの名前／値対のハッシュテーブルを特定する。

【００８０】表２は、LOSフォーマットの文法の一例を示す。

【００８１】

【表２】

制御オブジェクト	• value type-table$_{opt}$ name-table$_{opt}$	• h<name1;value1>¥t50System.Drawing.Color¥n1BackColor
値	• typed-value • untyped-value • typed-array-value • untyped-array-value • untyped-hashtable-value	• 50<red> • <red> • a50<red;blue;green> • a<red;blue;green> • h<name1;value1>
タイプされた値	• type-ref value-list-start value-ref value-list-end	• 50<red>
タイプされたアレイ値	• array-modifier type-ref value-list-start array-value-ref value-list-end	• a50<red;blue;green>
非タイプ値	• value-ref	• <red>
非タイプアレイ値	• array-modifier value-list-start array-value-ref value-list-end	• a<red;blue;green>
値リスト開始	• <	
値リスト終了	• >	
値リストセパレータ	• ;	
アレイモディファイア	• a	
ハッシュテーブルモディファイア	• h	
非タイプハッシュテーブル値	• hashtable-modifier value-list-start hashtable-value-ref value-list-end	• h<name1;value1>
値の参照	• string-value • bin-ref base64-persisted-object	• ¥"This is a string value.¥"
バイナリ参照	• escape-char b	• ¥b
アレイ値参照	• value$_1$;[value$_2$;[value$_n$]]	• red;blue;green
ハッシュテーブル値の参照	• name-ref$_1$;value$_1$ [value-list-separator name-ref$_2$;value$_2$ [value-list-separator name-ref$_n$;value$_n$]]	• <name1;value1; name2;value2; name3;value3>
名前の参照	• string-name-number • string-name	• 1 • BackColor
タイプの参照	• known-type-number • string-type-number • string-type	• 10 • 50 • System.Drawing.Color

【００８２】LOSフォーマットの使用方法の第１の例として、以下のエントリを考える。

h<name1;value1;name2;value with ¥; escaped ¥> ¥"characters¥">

これは、２対の名前／値のハッシュテーブルを規定し、ここでの値はすべてストリングである。第１の値は、"name1" という名で、ストリング"value1"と等しい。第２の値は、"name2"という名で、ストリング"value with ; escaped > "characters""と等しい。

【００８３】LOSフォーマットの使用方法の第２の例として、以下のエントリを考える。

h<control1;h<1;50<blue>;text;hello>control2;h<1;50<red>;control3;>¥t50System.Drawing.Color¥n1BackColor

これは、"control1"、"control2"および"control3"という３つのハッシュエントリを有するハッシュテーブルを規定する。第１のハッシュエントリは、"control1"という名であり、名前／値対である２つのエントリを有する子ハッシュテーブルを含む。control1ハッシュテーブルの第１のエントリ名は、この例の最後に規定されている"BackColor"という名を参照するためのインデックス「１」を用いる。第１のエントリ値は、"blue"と等しくインデックス「５０」で特定されこの例の最後近くに規定されている"System.Drawing.Color"タイプである。control1ハッシュテーブルの第２のエントリの名前／値対は、名前"text"および値"hello"を含んでいる。"control2"のハッシュエントリは、"BackColor"という名を参照するためのインデックス「１」をここでも用いる１つのハッシュエントリを有する子ハッシュテーブルを含んでいる。この値は、"red"と等しく、"System.Drawing.Color"タイプである。"control3"のハッシュエントリは空である。上述の実施形態と同様の方法で、階層をハッシュテーブルの入れ子によって特定するが、階層識別子の使用を含む他の階層の記述方法も本発明の範囲内であると考えられる。

【００８４】図１０は、本発明のある実施形態における

搬送可能状態構造を受け取り、そこに記憶された状態情報を制御オブジェクト階層の制御オブジェクトにロードするプロセスフローチャートを示す。図示した実施形態では、少なくとも１つのサーバ側制御オブジェクトの少なくとも１つのプロパティが変化していない限り、搬送可能状態構造はサーバから送信されない（したがってクライアントによってサーバに戻されない）。このようなものであるので、図示したフローチャートは、状態情報の少なくとも１つの要素が、受信した搬送可能状態構造に存在すると仮定している。別の実施形態では、たとえ搬送可能状態構造に状態情報がなくても、搬送可能状態構造がサーバとクライアントとの間を往復するものもある。このような実施形態では、搬送可能状態構造に状態情報がない場合、状態管理プロセスを打ち切るよう判断処理（図示せず）が作動し得る。

【００８５】受信処理１０００が、HTTPリクエストなどでクライアントから搬送可能状態構造を受信する。マーキング処理１００２は、サーバ階層のすべての制御オブジェクトの最初の状態指示を設定する。読出し処理１００４は、受信した搬送可能状態構造を読み出し、状態情報および対応するプロパティタイプを抽出する。

【００８６】チェック処理１００６は、搬送可能状態構造のインテグリティチェックを行う（例えば、##VIEWSTATEMAC値を用いて）。本発明のある実施形態では、チェック処理１００６において、（１）搬送可能状態構造から受信インテグリティコードを読み出す；（２）搬送可能状態構造に含まれる少なくとも状態情報からそれ自体のインテグリティコードを計算する；（３）受信インテグリティコードと新たに計算したインテグリティコードを比較して、搬送可能状態構造が、クライアントへの往復の間に変造されているかどうかを判断する。搬送可能状態構造が、変造されていると判断された場合、サーバ側コードは、ロード処理を打ち切る、状態情報なしで処理を継続する、またはエラーとすることなどによってこの例外を扱うよう応答することができる。

【００８７】検査処理１００８は、搬送可能状態構造に含まれるファイルチェックコードを検査する。本発明の実施形態では、検査処理１００８では、（１）##VIEWSTATEフィールドから受信したファイルチェックコードを読み出す；（２）少なくともサーバに記憶されたASP+リソースの内容からそれ自体のファイルチェックコードを計算する；（３）受信したファイルチェックコードと新たに計算したファイルチェックコードを比較して、搬送可能状態構造の状態情報は、サーバのASP+リソースの同じバージョンに対応するかどうかを判断する。検査処理１００８を用いて、サーバ上のASP+リソースが、クライアントへの搬送可能状態構造の往復の間に変化していないことを確認する。サーバ上のASP+リソースが変化した場合は、サーバは、受信した状態情報を破棄し、ロード処理を打ち切り、かつ／またはクライアントへエラーを送ったりする。ある実施形態では、最初の状態の制御オブジェクト階層を作成し、サーバがリクエストに対する処理を続ける前にロード処理を打ち切る。トラバース処理１０１０は、サーバ側制御オブジェクト階層のページ制御オブジェクトをトラバースする。

【００８８】構文解析処理１０１２は、入れ子型##VIEWSTATEフィールドの制御オブジェクトに対応する制御オブジェクト階層の状態情報を構文解析する。第１の反復において、対応する制御オブジェクトは、トラバース処理１０１０においてトラバースされたページ制御オブジェクトである。次の反復において、対応する制御オブジェクトは、ページ制御オブジェクトの子孫制御オブジェクトであり、トラバース処理１０２８によって所在を明らかにされる。構文解析処理１０１２は、状態値を抽出し、当初解析されたときはストリングフォーマットであったその値を所与のプロパティタイプへと変換することができる。ロード処理１０１４は、##VIEWSTATE値から構文解析された状態情報を制御オブジェクトのプロパティへロードする。指示処理１０１６は、現在の制御オブジェクトの変化状態指示を設定する。この指示は、後に、当初の状態から状態が変化した制御オブジェクトの状態だけを保存することにより、搬送可能状態構造のサイズを最小にするために用いることができる。あるいは、すべての状態情報を保存してもよい。ポストバックデータのハンドリング、ポストバックイベントのハンドリングおよび制御オブジェクト階層でのデータ結合処理の間、制御オブジェクトの状態になされた変化もまた、個々の制御オブジェクトの変化状態指示を設定したことになり得る。しかし、本発明のある実施形態では、データ結合処理のために設定されたどんなデータ変化指示も、そのプロパティの状態が搬送可能状態構造に記録されないようにリセットされ、それによって、搬送可能状態構造のサイズをさらに減少させる。このようなプロパティの状態は、図５のデータ結合処理５１２で更新される結合関係によって特定されたデータに依存する。

【００８９】判断処理１０１８において、現在の制御オブジェクトに追加の状態情報が存在する場合、処理は１０２２に進む。存在する場合、処理１０２２において、現在の制御オブジェクトにおける次の制御オブジェクトプロパティの状態情報を構文解析して、処理は１０１４に進む。判断処理１０１８において、現在の制御オブジェクトに追加の状態情報がない場合は、判断処理１０２０に進み、ここで、##VIEWSTATE値に他の制御オブジェクト状態情報がない場合は、処理１０２４に進む。判断処理１０２０において、状態が処理されていない追加の制御オブジェクトがある場合は、処理１０２６に進み、ここで、##VIEWSTATE値から次の制御オブジェクトの状態情報を構文解析する。トラバース処理１０２８では、処理１０２６で構文解析された新たな状態情報に対応する次の制御オブジェクトをトラバースする。その後、ロ

ード処理１０１４に進む。本発明のある実施形態では、図１０の反復構文解析処理が、非直列化処理を実行する。

【００９０】図１１は、本発明のある実施形態におけるクライアントへ送信する搬送可能状態構造へ制御オブジェクト階層の制御オブジェクトからの状態情報を保存するプロセスフローチャートを示す。初期化処理１１００は、搬送可能状態構造を初期化する。計算処理１１０２は、ASP+リソースの現在のバージョンに基づきファイルチェックコードを計算する。本発明の実施形態では、ファイルチェックコードは、ファイルの内容またはファイルの幾つかの特定されたコンポーネントを単にハッシュするなどの、ASP+リソース内容のハッシュアルゴリズム（例えば、MD5など）に基づく。ロード処理１１０４は、搬送可能状態構造に計算されたファイルチェックコードをロードする。トラバース処理１１０６は、制御オブジェクト階層のページオブジェクトをトラバースする。

【００９１】本発明の実施形態において、サーバ側制御オブジェクトは、制御オブジェクト（およびその子オブジェクト）のプロパティ値を搬送可能状態構造に保存すべきかどうかを示す"MaintainState"プロパティを用いて作成することができる。保存する必要がない場合、トラバース処理１１０６は、このプロセスでサーバ側制御オブジェクトおよびその子オブジェクトをスキップすることができる。そうでなく、制御オブジェクトの状態を"MaintainState"プロパティに従い保持すべきなら、トラバース処理１１０６は制御オブジェクトをトラバースする。

【００９２】判断処理１１０８により、処理は、読出し処理１１１５に進む。ここでは、制御オブジェクトプロパティから新たな状態値とそのタイプを読み出す。処理１１１０は、新たな状態値とそのタイプを記録し、処理１１１１に進む。この説明のために、現在の制御オブジェクト内にプロパティが存在しない場合、プロパティを##VIEWSTATEフィールドの"null"値で表す。現在の制御オブジェクトのプロパティが、依然として処理１１０８での当初の状態のままであれば、判断処理１１１１に進む。処理１１１１において現在の制御オブジェクト内に別のプロパティが存在する場合は、判断処理１１０８に進む。

【００９３】処理１１１１において現在の制御オブジェクト内に他のプロパティが存在しない場合は、判断処理１１１２に進み、ここで、別の制御オブジェクトが階層に存在する場合はトラバース処理１１１４に進む。トラバース処理１１１４は、制御オブジェクト階層内の次の制御オブジェクトをトラバースし、判断処理１１０８に進み制御オブジェクトのプロパティにアクセスする。処理１１１２において、他の制御オブジェクトが階層に存在しない場合は、先の処理で得られた新たな状態情報（状態値タイプを含む）を直列化し、それらを搬送可能状態構造に記憶する直列化処理１１１６に進む。処理１１１８は、（MD5アルゴリズムを用いるなどして）インテグリティコードを計算し、処理１１２０は、例えば##VIEWSTATEMACフィールドでインテグリティコードを搬送可能状態構造に記憶する。インテグリティコード処理およびフィールドチェック処理は、任意であり、本発明の別の実施形態では省くことができる。処理１１２２は、例えば、制御オブジェクト階層によって生成されたレンダリングされたHTMLコードを含むHTTPリクエストなどでクライアントへ搬送可能状態構造を送信する。

【００９４】本明細書中の本発明の実施形態は、１つ以上のコンピュータシステムの論理ステップとして実行される。本発明の論理処理は、（１）１つ以上のコンピュータシステムで実行処理されるプロセッサ実行ステップシーケンスとして、（２）１つ以上のコンピュータシステム内の相互接続された機械モジュールとして実行される。実行は選択の問題であり、本発明を実行するコンピュータシステムの性能要件に依存する。したがって、本明細書に記載の本発明の実施形態を構成する論理処理は、処理、ステップ、オブジェクトまたはモジュールと様々に表現することができる。

【００９５】上記の明細書、実施例およびデータは、本発明の実施形態の構造および使用の完全な説明を提供している。本発明は、本発明の精神および範囲から逸脱することなく多数の形態で実施することができるので、本発明は添付の請求項にある。

【００９６】

【発明の効果】

【図面の簡単な説明】

【図１】　本発明のある実施形態におけるクライアントに表示するウェブページコンテンツを動的に生成するウェブサーバを示す。

【図２】　本発明のある実施形態におけるサーバ側制御オブジェクトを用いてクライアント側ユーザインタフェース要素の処理およびレンダリングのための処理のフローチャートを示す。

【図３】　本発明のある実施形態で用いるウェブサーバのモジュールの一例を示す。

【図４】　本発明のある実施形態を実行するのに有用なシステムの一例を示す。

【図５】　本発明のある実施形態におけるページオブジェクトの処理を表すプロセスフローチャートを示す。

【図６】　本発明のある実施形態における動的コンテンツファイル（例えば、ASP+リソース）の部分の一例を示す。

【図７】　図６の動的コンテンツファイルの部分の一例に応じて、１つ以上のサーバ側制御オブジェクトによって生成された結果コードを示す。

【図８】　図６のASP+リソースに対応する制御オブジェ

クト階層および図７の##VIEWSTATEフィールド値を示す。

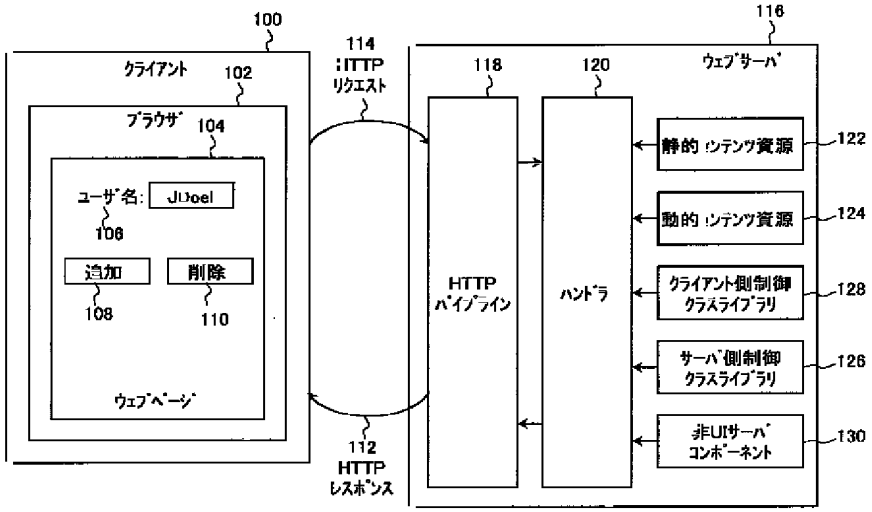
【図９】　図７の##VIEWSTATEフィールド値の入れ子版を示す。

【図１０】　本発明のある実施形態における搬送可能状態構造を受け取り、それに格納された状態情報を制御オブジェクト階層の制御オブジェクトへロードするプロセスフローチャートを示す。

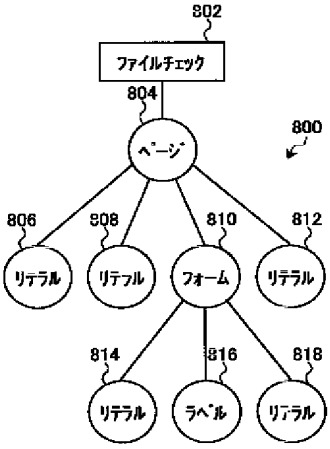
【図１１】　本発明のある実施形態における制御オブジェクト階層の制御オブジェクトからの状態情報を搬送可能状態構造へ保存しクライアントへ送信するプロセスフローチャートを示す。

【符号の説明】

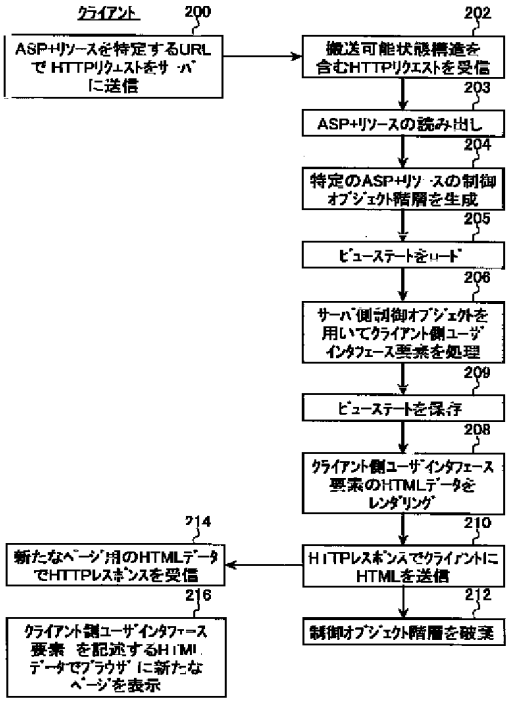
【図１】



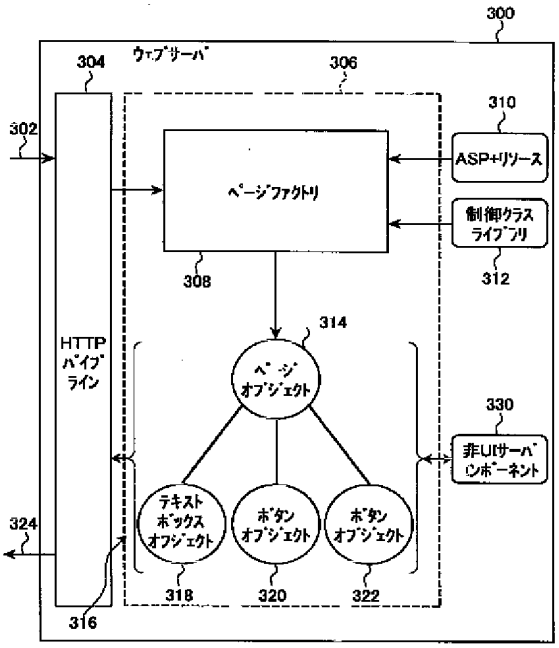
【図８】



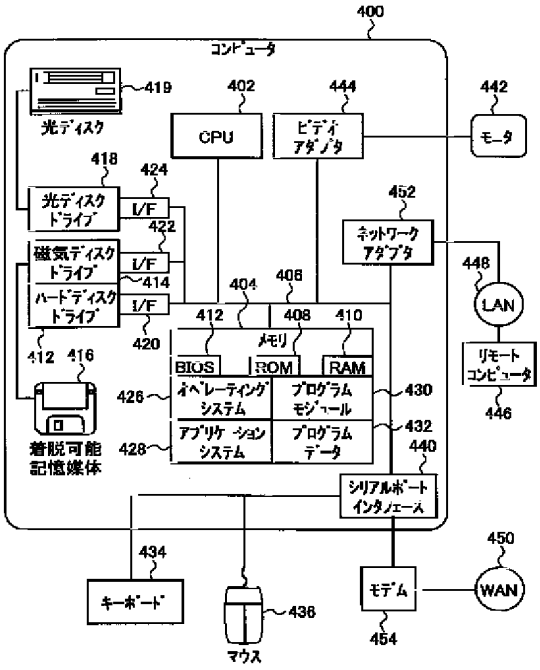
【図２】



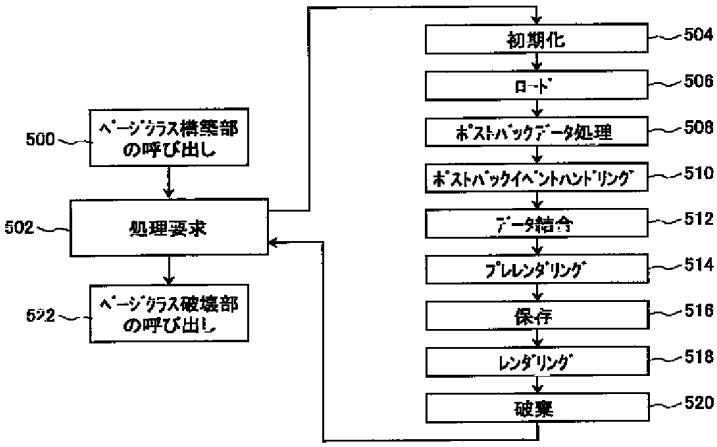
【図３】



【図４】



【図５】



【図６】

600

```
1    <html>
2         <script runat=server>
3              Overrides Sub Load()
4                   If (IsFirstLoad) Then
5                        Message.Text = "You've never visited this page before!"
6                   Else
7                        Message.Text = "Last visited: "& State("LastAccessed")
8                   End If
9              Overrides Sub Save()
10                  State("LastAccessed") = Now
11             End Sub
12        </script>

13        <body>
14             <form runat=server>
15                  <span id="Message" runat=server/>
16             </form>
17        </body>
18   </html>
```

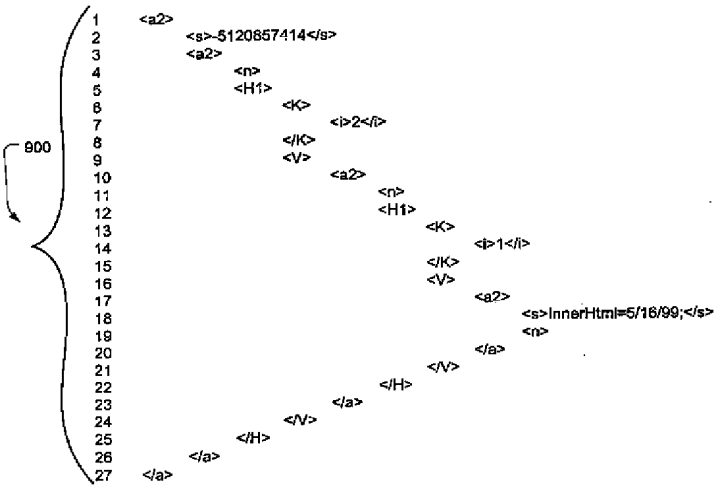
【図７】

700

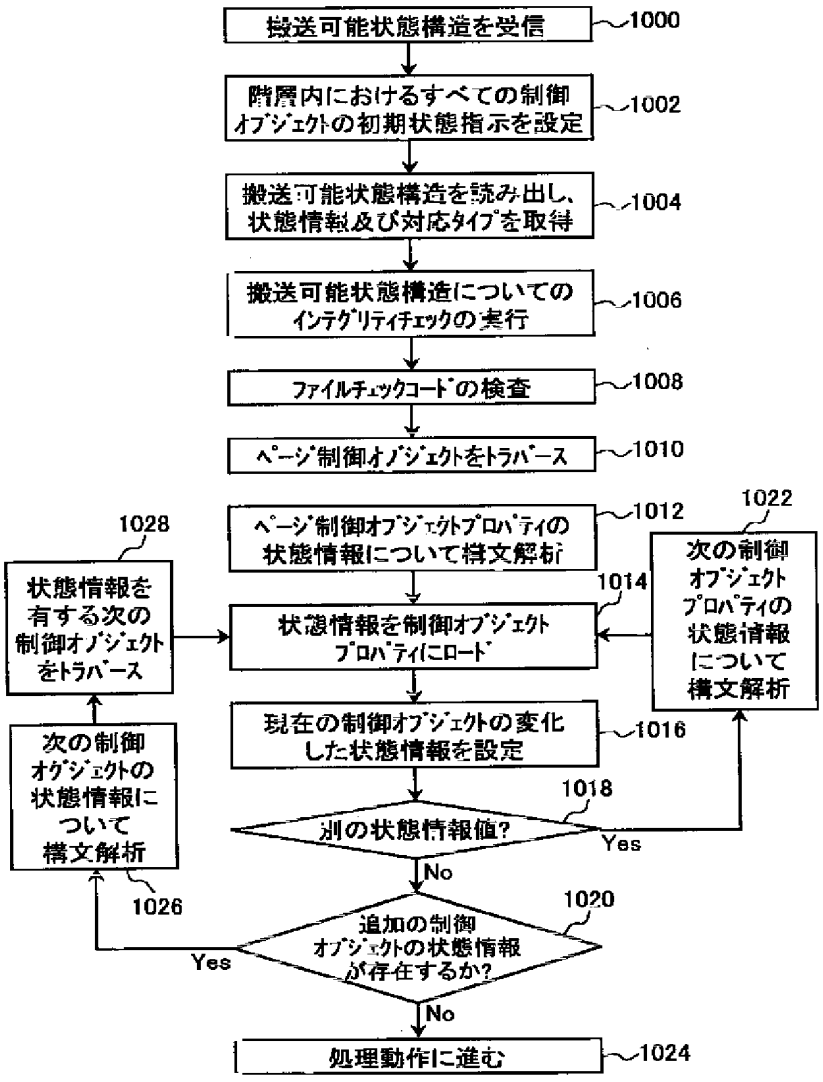
```
1    <form>
2         <input type=hidden name="__VIEWSTATE"value="<a2><s>-5120857414</s><a2>
     <n><H1><K><i>2</i></K><V><a2><n><H1><K><i>1</i></K><V><a2><s>InnerHtml=5/16/99;
     </s><n></a></V></H></a></V></H></a></a>">
3         <input type=hidden name ="__VIEWSTATEMAC" value="43434333433">

4         <span id "Message">Last Post Back: 5/6/99</span>
5    </form>
```

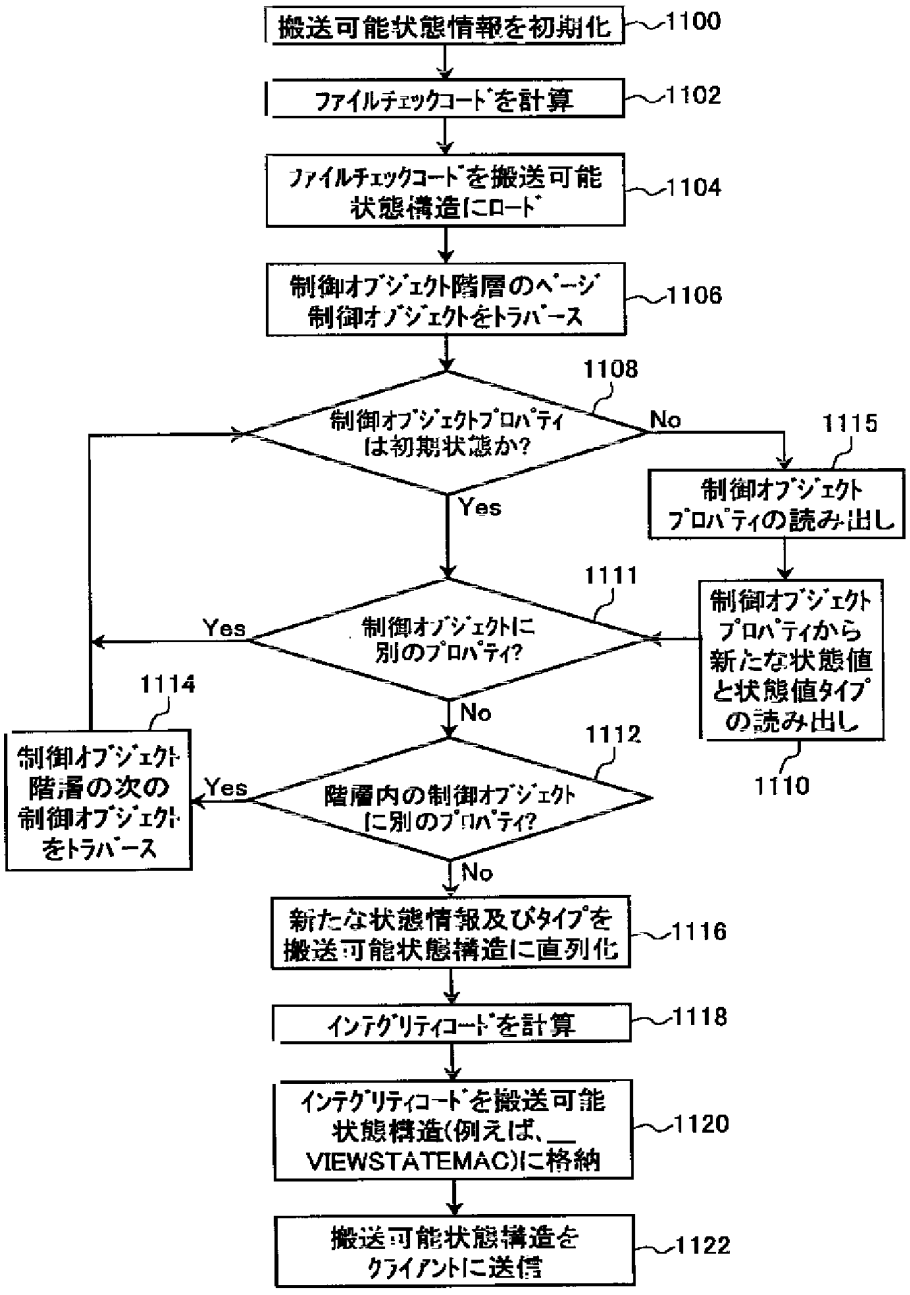
【図９】


900
1 <a2>
2 <s>-5120857414</s>
3 <a2>
4 <n>
5 <H1>
6 <K>
7 2
8 </K>
9 <V>
10 <a2>
11 <n>
12 <H1>
13 <K>
14 1
15 </K>
16 <V>
17 <a2>
18 <s>InnerHtml=5/16/99;</s>
19 <n>
20
21 </V>
22 </H>
23
24 </V>
25 </H>
26
27

【図１０】


搬送可能状態構造を受信
1000
階層内におけるすべての制御オブジェクトの初期状態指示を設定
1002
搬送可能状態構造を読み出し、状態情報及び対応タイプを取得
1004
搬送可能状態構造についてのインテグリティチェックの実行
1006
ファイルチェックコードの検査
1008
ページ制御オブジェクトをトラバース
1010
ページ制御オブジェクトプロパティの状態情報について構文解析
1012
状態情報を制御オブジェクトプロパティにロード
1014
現在の制御オブジェクトの変化した状態情報を設定
1016
別の状態情報値?
1018
Yes
No
追加の制御オブジェクトの状態情報が存在するか?
1020
Yes
No
処理動作に進む
1024
状態情報を有する次の制御オブジェクトをトラバース
1028
次の制御オブジェクトの状態情報について構文解析
1026
次の制御オブジェクトプロパティの状態情報について構文解析
1022

【図１１】



【手続補正書】
【提出日】平成１３年6月１２日（２００１．６．１２）
【手続補正１】
【補正対象書類名】明細書
【補正対象項目名】００９６
【補正方法】変更
【補正内容】
【００９６】
【発明の効果】以上のように本発明によれば、状態管理を行わないクライアント／サーバモデルにおいて、サーバ側制御オブジェクトの状態情報が２つのHTTPリクエストの間はサーバ側に保持されるのではなく、搬送可能な状態構造でクライアントとサーバ間を搬送されることになることから、適切なサーバ側制御オブジェクトによっ

て、ウェブページに表示するクライアント側ユーザインタフェース要素を処理および生成することができ、動的コンテンツが対象となるウェブページの開発をより容易に行うことが可能となる。

【手続補正２】
【補正対象書類名】明細書
【補正対象項目名】符号の説明
【補正方法】変更
【補正内容】
【符号の説明】
１００　クライアント
１０２　ブラウザ
１０４　ウェブページ
１０６　ユーザ名
１０８　追加
１１０　削除
１１２　HTTPレスポンス
１１４　HTTPリクエスト
１１６、３００　ウェブサーバ
１１８、３０４　HTTPパイプライン
１２０　ハンドラ
１２２　静的コンテンツ資源
１２４　動的コンテンツ資源
１２６　サーバ側制御クラスライブラリ
１２８　クライアント側制御クラスライブラリ
１３０、３３０　非UIサーバコンポーネント
３０８　ページファクトリ
３１０　ASP+リソース
３１２　制御クラスライブラリ
３１４　ページオブジェクト
３１８　テキストボックスオブジェクト
３２０、３２２　ボタンオブジェクト
４００　コンピュータ
４０４　メモリ
４１２　ハードディスクドライブ
４１４　磁気ディスクドライブ
４１６　着脱可能記憶媒体
４１８　光ディスクドライブ
４１９　光ディスク
４２０、４２２、４２４　Ｉ／Ｆ
４２６　オペレーティングシステム
４２８　アプリケーションプログラム
４３０　プログラムモジュール
４３２　プログラムデータ
４３４　キーボード
４３６　マウス
４４０　シリアルポートインタフェース
４４２　モニタ
４４６　リモートコンピュータ
４５２　ネットワークアダプタ
４５４　モデム
８０２　ファイルチェック
８０４　ページ
８０６、８０８、８１２、８１４、８１８　リテラル
８１０　フォーム
８１６　ラベル

フロントページの続き

(51)Int.Cl.[7]	識別記号	ＦＩ		(参考)
Ｇ０６Ｆ　17/30	１４０ １８０	Ｇ０６Ｆ　17/30	１４０ １８０Ｅ	

(72)発明者　クーパー、ケネス　ビー.
アメリカ合衆国、98199　ワシントン州　シアトル、ウエスト　ブレイン　3410
(72)発明者　アンダース、マーク　ティー.
アメリカ合衆国、98007　ワシントン州　ベルビュー、エヌイー　12ス　プレイス　14425
(72)発明者　ギャスリー、スコット　ディー.
アメリカ合衆国、98052　ワシントン州　レッドモンド、アパートメント　ユー　2102、エヌイー　90ス　ストリート　17786
(72)発明者　エボ、デイビッド　エス.
アメリカ合衆国、98052　ワシントン州　レッドモンド、アパートメント　ジェイ　139、アボンデイル　ロード　10909
(72)発明者　ピータース、テッド　エー.
アメリカ合衆国、98119　ワシントン州　シアトル、８ス　アベニュー　ウエスト　2431
(72)発明者　ミレット、スティーブン　ジェイ.
アメリカ合衆国、98026　ワシントン州　エドモンズ、オリンピック　ビュー　ドライブ　8412

Ｆターム(参考)　5B075 KK07 KK20 KK33 ND20 ND23
　　ND35 QT03 UU40
　　5B085 AC13 BG07
　　5B098 AA10 GA01 GC11 GC14

【外国語明細書】

1 Title of Invention

STATE MANAGEMENT OF SERVER-SIDE CONTROL OBJECTS

2 Claims

1. A method for managing a state of a server-side control object corresponding to a client-side user interface element incorporated in a web page displayed on a client, the method comprising:

creating the server-side control object in the control object hierarchy to process the client-side user interface element;

receiving from the client a transportable state structure including state information indicating a state value for at least one server-side control object in the control object hierarchy,

extracting the state information from the transportable state structure; and

loading the state value from the state information into property of the server-side control object, if the state value is associated with the server-side control object.

2. The method of claim 1 wherein the extracting operation comprises:

extracting from the transportable state structure a state value associated with a property of the at least one server-side control object; and

identifying a hierarchical location of the server-side control object within the control object hierarchy based on hierarchical information within the transportable state structure.

3. The method of claim 2 wherein the loading operation comprises:

locating the server-side control object within the control object hierarchy based on the hierarchical location identified in the identifying operation;

traversing the hierarchical location of the server-side control object within the control object hierarchy; and

storing the state value into the property of the server-side control object.

4. The method of claim 1 wherein the transportable state structure further includes a received integrity code, and further comprising:

reading the received integrity code from the transportable state structure;

calculating a calculated integrity code from the state information included in

the transportable state structure; and

comparing the received integrity code with the calculated integrity code to determine whether the transportable state structure is valid.

5. The method of claim 1 further comprising:

initializing a property of the server-side control object to have an initial state;

setting an indication to represent that the state of the property of the server-side control object is unchanged from the initial state; and

changing the indication to represent that the property of the server-side control object has changed from the initial state, if the property was loaded with the state value from the transportable state structure.

6. The method of claim 5 further comprising:

setting the indication to indicate that the property of the server-side control object is unchanged from the initial state, if the property is data bound to a field of a server-side datastore.

7. The method of claim 5 further comprising:

traversing each server-side control object in the control object hierarchy; and

storing the state information of the server-side control object into the transportable state structure for transmission back to the client, if the indication represents that the state information of the server-side control object has changed from its initial state.

8. The method of claim 7 further comprising:

transmitting the transportable state structure to the client in a response with authoring language data defining the web page.

9. The method of claim 7 further comprising:

calculating an integrity code from the state information include in the transportable state structure, responsive to the operation of storing the state information;

storing the integrity code in the transportable state structure; and

transmitting the transportable state structure to the client in a response with

authoring language data defining the web page.

10. The method of claim 1 further comprising:

generating authoring language code defining at least a portion of the web page from the at least one server-side control object in the control object hierarchy.

11. The web page defined by the authoring language code and being interpretable by a browser executing on a client computer system coupled to a web server, the authoring language code being substantially generated at the web server by the method of claim 10, wherein the authoring language code includes the transportable state structure storing state information for at least one server-side control object in the control object hierarchy.

12. A computer program storage medium readable by a computer system and encoding a computer program for executing a computer process managing a state of a server-side control object corresponding to a client-side user interface element incorporated in a web page displayed on a client, the computer process comprising:

creating the server-side control object in the control object hierarchy to process the client-side user interface element;

receiving from the client a transportable state structure including state information indicating a state value for at least one server-side control object in the control object hierarchy,

extracting the state information from the transportable state structure; and

loading the state value from the state information into property of the server-side control object, if the state value is associated with the server-side control object.

13. A computer data signal embodied in a carrier wave by a computing system and encoding a computer program for executing a computer process managing a state of a server-side control object corresponding to a client-side user interface element incorporated in a web page displayed on a client, the computer process comprising:

creating the server-side control object in the control object hierarchy to process the client-side user interface element;

receiving from the client a transportable state structure including state information indicating a state value for at least one server-side control object in the control object hierarchy,

extracting the state information from the transportable state structure; and

loading the state value from the state information into property of the server-side control object, if the state value is associated with the server-side control object.

14. A computer program product encoding a computer program for executing in a computer system a computer process for managing a state of a plurality of server-side control objects created in a control object hierarchy and corresponding to a plurality of client-side user interface elements incorporated in a web page displayed on a client, the computer process comprising:

receiving from the client a transportable state structure including state information associated with one or more server-side control objects in the control object hierarchy;

deserializing the state information to extract a state value, an associated property data type, and hierarchical information for a property of a server-side control object;

locating the server-side control object within the control object hierarchy based on the hierarchical information; and

loading the state value into the property of the server-side control object.

15. The computer program product of claim 14 wherein the computer process further comprises:

converting the state value have the associated property data type prior to the loading operation.

16. The computer program product of claim 14 wherein the computer process further comprises:

initializing the property of the server-side control object to have an initial state;

setting an indication to represent that the property of the server-side control object is unchanged from the initial state; and

changing the indication to represent that the property of the server-side control object has changed from the initial state, if the property was loaded with the state value from the transportable state structure.

17. The computer program product of claim 15 wherein the loading operation comprises traversing one or more of the server-side control objects in the control object hierarchy, and for each server-side control objects, the loading operation further comprises:

extracting a property value from the server-side control object associated with the indication representing that the state of the server-side control object has changed from its initial state; and

serializing the property value into the transportable state structure with the property values from other server-side control objects in the control object hierarchy for transmission back to the client.

18. The computer program product of claim 16 wherein the loading operation further comprises:

extracting a property data type from the server-side control object associated with the property value; and

serializing the property data type into the transportable state structure with the property value of the server-side control object for transmission back to the client.

19. The computer program product of claim 16 wherein the loading operation further comprises:

storing into the transportable state structure hierarchical information relating to the server-side control object in the control object hierarchy for transmission to the client.

20. The computer program product of claim 17 wherein the computer process further comprises:

transmitting the transportable state structure to the client in a response with authoring language data defining a web page.

21. The computer program product of claim 17 wherein the computer process further comprises:

calculating an integrity code from the state information stored in the transportable state structure, responsive to the serializing operation;

storing the integrity code in the transportable state structure; and

transmitting the transportable state structure to the client in a response with authoring language data defining a web page.

22. The computer program product of claim 14 wherein the transportable state structure further includes a received integrity code, and further comprising:

reading the received integrity code from the transportable state structure;

calculating a calculated integrity code from the state information include in the transportable state structure; and

comparing the received integrity code with the calculated integrity code to determine if the transportable state structure is valid.

23. A handler system executed in a server computer for processing a request including a transportable state structure storing a state value and a data type from a client computer, the handler system comprising:

a hierarchy of server-side control objects created by the handler and associated with client-side user interface elements, at least one of the server-side control objects including a property;

a loader module storing the state value received in the transportable state structure into the property of the server-side control object, based on the data type; and

a saver module storing the property and the data type into a transportable state structure.

24. The handler system of claim 23 further comprising:

a reader module extracting the state value, associated hierarchical information, and the data type from transportable state structure; and

a traversal module locating the control object within the control object hierarchy, based on the associated hierarchical information.

3 Detailed Description of Invention

Technical Field

The invention relates generally to a web server framework, and more particularly to managing the state of server-side control objects that process client-side user interface elements of a web page.

Background of the Invention

A typical web browser receives data from a web server that defines the appearance and rudimentary behavior of a web page for display on a client system. In a typical scenario, a user specifies a Uniform Resource Locator ("URL"), which is a global address of a resource on the World Wide Web, to access a desired web site. Generally, the term "resource" refers to data or routines that can be accessed by a program. An example URL is "http://www.microsoft.com/ms.htm". The first part of the example URL indicates a given protocol (i.e., "http") to be used in the communication. The second part specifies the domain name (i.e., "www.microsoft.com") where the resource is located. The third part specifies the resource (i.e., a file called "ms.htm") within the domain. Accordingly, a browser generates an HTTP (HyperText Transport Protocol) request associated with the example URL to retrieve the data associated with ms.htm file within the www.microsoft.com domain. A web server hosting the www.microsoft.com site receives the HTTP request and returns the requested web page or resource in an HTTP response to the client system for display in the browser.

The "ms.htm" file of the example above includes static HTML (HyperText Markup Language) code. HTML is a plain-text authoring language used to create documents (e.g., web pages) on the World Wide Web. As such, an HTML file can be retrieved from a web server and displayed as a web page in a browser. Using HTML, a developer can, for example, specify formatted text, lists, forms, tables, hypertext links, inline images and sounds, and background graphics for display in

the browser to present the rich graphical experience that users have come to expect while viewing information from the Internet. An HTML file, however, is a static file that does not inherently support dynamic generation of web page content.

If dynamic content is to be displayed, such as a change stock price or traffic information, a server-side application program is generally developed to handle the more complex client-server interaction. The server-side application program processes an HTTP request and generates the appropriate HTML code for transmission to the client in an HTTP response. An exemplary HTTP request may include parameters, such as data in a query string or data from web-based forms. As such, a server-side application program can process the parameters and dynamically generate HTML code in an HTTP response to the client. An exemplary server-side application program may dynamically generate documents containing appropriate HTML code using a sequence of one or more formatted text write operations to a memory structure. Thereafter, the resulting document is transmitted to a client system in an HTTP response, where it is displayed as a web page in the browser.

Developing a server-side application program can be a complex task requiring not only familiarity with normal HTML coding that is used to layout a web page, but also with programming basics, including one or more programming languages, (e.g., C++, Perl, Visual Basic, or Jscript). Web page designers, on the other hand, are frequently graphics designers and editors, who may lack programming experience. Furthermore, simplifying complex web page development can speed the development of new web content by any developer. Generally, development of a custom server-side application program also requires tremendous effort, so much, in fact, that developers are often disinclined to attempt it. It is desirable, therefore, to provide a development framework that allows a developer to dynamically create and process a web page with minimal programming.

One approach to minimize the programming requirements of dynamic web page generation has been the Active Server Page (ASP) framework, provided by Microsoft Corporation. An ASP resource typically includes Visual Basic or Jscript code, for example, to process an HTTP request that specifies the ASP resource as the desired resource and, thereafter, to generate the resulting HTML code in a HTTP response to the client. Furthermore, an ASP resource may reference pre-developed

or third party client-side library components (e.g., client-side "ACTIVEX" controls) instead of requiring that the page developer write all components from scratch. However, in the current server-side application frameworks, the programming required to dynamically manage client-side user interface elements (e.g., text boxes, list boxes, buttons, hypertext links, images, sounds, etc.) within server-side applications can still require sophisticated programming skills and considerable effort. An unanswered problem exists in property encapsulating programming required to process user interface elements, so as to allow the web page developer to focus on other aspects of the web page..

Server-side processing of client-side user interface elements, however, may involve complex state management issues, which would typically fall to the server-side applications program. The state of a server-side processing module corresponding to the client-side user interface element represents the server-side module's properties and configuration. A stateless client/server model, however, dictates that communications between client and server the server does not maintain the state of processes occurring between the client and the server, because the communications connection between the client and server may disappear unexpectedly.

Summary of the Invention

In accordance with the present invention, the above and other problems are solved by providing encapsulated state management for server-side processing of client-side user interface elements in which the server does not maintain the state between requests. The operation of processing the client-side user interface element may include a state management operation, which relates to the state (i.e., the "viewstate") of one or more server-side control objects in a control object hierarchy. A server-side control object processes and generates a client-side user interface element for display on a web page. A hierarchy of server-side control objects may also cooperate to generate the resulting authoring language code, such as standard HTML, for display of a web page on a client. The client can be, for example, any browser that supports standard HTML or another authoring language.

To satisfy the stateless client/server model, state information of server-side

control objects may be transported between the client and the server in a transportable state structure, rather than remaining on the server between HTTP requests. When the server receives an HTTP request from the client, the server extracts the state information from a transportable state structure from the HTTP request and distributes the state information to the appropriate individual control objects in the server-side hierarchy. The integrity of the transportable state structure may also be verified using a coded structure associated with the transportable state structure and generated from the state information.

A method for managing a state of a server-side control object corresponding to a client-side user interface element incorporated in a web page displayed on a client is provided. The server-side control object is created in the control object hierarchy to process the client-side user interface element. A transportable state structure is received from the client. The transportable state structure includes state information indicating a state value for at least one server-side control object in the control object hierarchy. The state information is extracted from the transportable state structure. The state value from the state information is loaded into property of the server-side control object, if the state value is associated with the server-side control object.

A computer program product for managing a state of a plurality of server-side control objects created in a control object hierarchy and corresponding to a plurality of client-side user interface elements incorporated in a web page displayed on a client is provided. A transportable state structure is received from the client includes state information associated with one or more server-side control objects in the control object hierarchy. The state information is serialized to extract a state value, an associated property data type, and hierarchical information for a property of a server-side control object. The server-side control object is located within the control object hierarchy based on the hierarchical information. The state value is loaded into the property of the server-side control object.

An article of manufacture is provided as a computer program product. An embodiment of a computer program product in accordance with the present invention includes a computer program storage medium readable by a computer system and encoding a computer program for executing a computer process managing a state of a server-side control object corresponding to a client-side user interface element incorporated in a web page displayed on a client. An alternative embodiment of a computer program product in accordance with the present invention includes a computer data signal embodied in a carrier wave by a computing system and encoding a computer program managing a state of a server-side control object corresponding to a client-side user interface element incorporated in a web page displayed on a client. A product generated by a process of the present invention is provided as authoring language code, containing a transportable state structure, transmitted to a client and interpreted by a browser on the client.

Detailed Description of the Invention

In an embodiment of the present invention, web page content is dynamically generated on a web server for display on a client. The client and the web server communicate across a network, for example, using HTTP requests and HTTP responses. As such, the web server generates web page content, which may be in the form of HTML code, and transmits the content to the client, which can display the web page in a browser. Server-side control objects, which logically correspond to individual user interface elements of the web page, are created on the web server to process and generate the web page content to be used by a client-side browser to display and process a web page. The server-side control objects are declared in a dynamic content file, such as an ASP+ resource, which is processed by a page factory to create a hierarchy of server-side control objects. The control objects in the hierarchy cooperate to process the request received from the client and then generate resulting web page content for transmission to the client, before the control objects in the hierarchy are terminated.

A page object may be instantiated as the top level of the control object hierarchy in an embodiment of the present invention. A page object, which is also a control object, typically contains one or more child control objects, and each child control object can contain one or more child control objects of its own to extend into a hierarchy of multiple levels. The page objects and descendent control objects execute a sequence of operations to process and generate the web content that corresponds to client-side user interface elements.

One of the operations in this sequence includes a load operation that deserializes the state information, which is received in a transportable state structure included in the request from the client, into the appropriate control object or objects in the hierarchy. A traversal operation of this load operation may walk through the control object hierarchy, following hierarchical information in the transportable state structure, to locate appropriate control objects in the hierarchy. In an embodiment of

the present invention, the received state information includes only the states of control objects that have changed from their initial states (i.e., the state information includes only differential data). Alternative embodiments may include all states of all control objects in the hierarchy, whether changed or not, or some other combination of state information and hierarchical information for one or more server-side control objects.

Another operation of the sequence includes a save operation that serializes the state information from control objects having changed states. The state information is added to a transportable state structure for transmission to the client. The client may return the transportable state structure to the web server on a subsequent request to the web server. Upon receipt of the transportable state structure, the control object hierarchy is recreated and the load operation restores the hierarchy's state to that of the previous response.

In an alternative embodiment, the transportable state structure may not be transmitted to the client. Instead, the transportable state structure may remain in the server or be transferred to another resource location (e.g., another server). For example, in the case of load balancing, a first server may communicate with a browser in a first HTTP transaction, but a second server may communicate with the browser in a second HTTP transaction. Accordingly, the transportable state structure can be communicated from the first server to the second server to process the second HTTP transaction, thereby bypassing the client.

FIG. 1 illustrates a web server for dynamically generating web page content for display on a client in an embodiment of the present invention. A client 100 executes a browser 102 that displays a web page 104 on a display device of the client 100. The client 100 may include a client computer system having a display device, such as a video monitor. An "INTERNET EXPLORER" browser, marketed by Microsoft Corporation, is an example of a browser 102 in an embodiment of the present invention. Other exemplary browsers include without limitation "NETSCAPE NAVIGATOR" and "MOSAIC". The exemplary web page 104 incorporates a text box control 106 and two button controls 108 and 110. The browser 102 may receive HTML code in the HTTP response 112 from a web server 116 and displays the web page as described by the HTML code. Although HTML is described with reference to one embodiment, other authoring languages, including

without limitation SGML (Standard Generalized Markup Language), XML (eXtensible Markup Language), and WML (Wireless Markup Language), which is an XML-based markup language, designed for specifying the content and user interfaces of narrowband wireless devices, such as pagers and cellular phones, are contemplated within the scope of the present invention. Furthermore, although standard HTML 3.2 is primarily disclosed herein, any version of HTML is supportable within the scope of the present invention.

The communications between the client 100 and the web server 116 may be conducted using a sequence of HTTP requests 114 and HTTP responses 112. Although HTTP is described with reference to one embodiment, other transport protocols, including without limitation S-HTTP, are contemplated within the scope of the present invention. On the web server 116, an HTTP pipeline module 118 receives an HTTP request 114, resolves the URL, and invokes an appropriate handler 120 for processing the request. In an embodiment of the present invention, a plurality of handlers 120 for handling different types of resources is provided on the web server 116.

For example, if the URL specifies a static content file 122, such as an HTML file, a handler 120 accesses the static content file 122 and passes the static content file 122 back through the HTTP pipeline 118 for communication to the client 100 in an HTTP response 112. Alternatively, in an embodiment of the present invention, if the URL specifies a dynamic content file 124, such as an ASP+ resource, a handler 120 accesses the dynamic content file 124, processes the contents of the dynamic content file 124, and generates the resulting HTML code for the web page 104. In an embodiment of the present invention, the resulting HTML code includes standard HTML 3.2 code. Generally, a dynamic content file is a server-side declaration datastore (e.g., an ASP+ resource) that can be used to dynamically generate the authoring language code that describes a web page to be displayed on a client. The HTML code for the web page is then passed through the HTTP pipeline 118 for communication to the client 100 in an HTTP response 112.

During its processing, a handler 120 can also access libraries of pre-developed or third party code to simplify the development effort. One such library is a server-side class control library 126, from which the handler 120 can instantiate server-side control objects for processing user interface elements and generating the

resultant HTML data for display of a web page. In an embodiment of the present invention, one or more server-side control objects map to one or more user interface elements, visible or hidden, on the web page described in the dynamic content file 124.

A second library, in contrast, is a client-side control class library 128, such as a library including "ACTIVEX" components from Microsoft Corporation. An "ACTIVEX" control is a COM ("Component Object Model") object that follows certain standards in how it interacts with its client and other components. A client-side "ACTIVEX" control, for example, is a COM-based component that can be automatically downloaded to a client and executed by a web browser on the client. Server-side ACTIVEX components (not shown) are COM-based components that may be implemented on a server to perform a variety of server-side functions, such as providing the server-side functionality of a stock price look-up application or database component. A more detailed discussion of ACTIVEX can be found in "Understanding ACTIVEX and OLE", David Chappell, Microsoft Press, 1996.

In contrast to "ACTIVEX" controls, a server-side control object in an embodiment of the present invention, being specified in a dynamic content resource 124, logically corresponds to a user interface element that is incorporated in the web page on the client. The server-side control object can also generate valid HTML code that can include, for example, an HTML tag and a locator referencing a given client-side "ACTIVEX" control. If the browser already has the code for the client-side "ACTIVEX" control within its storage system, the browser executes the "ACTIVEX" control within the web page on the client. Otherwise, the browser downloads the code for the "ACTIVEX" control from the resource specified by the locator and then executes the "ACTIVEX" control within the web page on the client. A server-side control object in an embodiment of the present invention can also raise events to a server-side "ACTIVEX" control used to implement a stock look-up application on the server.

A handler 120 also has access to one or more non-user-interface server components 130 that execute on the web server 116 or on another accessible web server. A non-user-interface server component 130, such as a stock price look-up application or database component, may be referenced in or associated with a dynamic content file 124 that is processed by a handler 120. Sever-side events

raised by the control objects declared in the dynamic content file 124 may be processed by server-side code, which calls appropriate methods of the non-user-interface server component 130. As a result, the processing provided by the server-side control objects simplifies the programming of the non-user-interface server component 130 by encapsulating the processing and generation of the user interface elements of a web page, which allows the developer of the non-user-interface server component 130 to concentrate on the specific functionality of the application, rather than on user interface issues.

FIG. 2 illustrates a flow diagram of operations for processing and generating client-side user interface elements using server-side control objects in an embodiment of the present invention. In operation 200, the client transmits an HTTP request to the server. The HTTP request includes a URL that specifies a resource, such as an ASP+ resource. In operation 202, the server receives the HTTP request and invokes the appropriate handler for processing the specified resource. The HTTP request includes a transportable state structure including state information, and optionally hierarchical information, associated with one or more server-side control objects, although the first HTTP request to a given page typically does not include an transportable state structure because no state changes have occurred at the server for the given page. In addition, the transportable state structure may include property data type information, and an integrity code to assist the server in validating the state information (i.e., verify that the state information was not corrupted at the client). Operation 204 generates a server-side control object hierarchy based on the contents of the specified dynamic content file (e.g., the ASP+ resource). The ASP+ resource is read in operation 203. Operation 205 loads state information received in the transportable state structure to the appropriate server-side control objects in the hierarchy to restore the control objects to their previous state.

In operation 206, the server-side control objects of the control object hierarchy perform one or more of the following operations: Postback event handling, postback data handling, state management, and data binding. Postback events and data (collectively "postback input") from user interface elements are communicated from the client to the server for processing. A postback event, for example, may include without limitation a "mouse click" event from a client-side

button element or a "data change" event from a client-side textbox element that is communicated to the server. Postback data, for example, may include without limitation text entered by a user in a text box element or an index of an item selected from a drop-down box.

Operation 209 saves the property values, associated type information, and hierarchical information into a transportable state structure for transmission to the client. In operation 208, each server-side control object in the hierarchy is called to generate (or render) authoring language data, such as HTML code, for display of client-side user interface elements in the web page. Note that, although the term "render" may be used to described the operation of displaying graphics on a user interface, the "render" is also used herein to describe the operation of generating authoring language data that can be interpreted by a client application, such as a browser, for display and client-side functionality. Rendering operation 208 also generates the authoring language data representing the transportable state structure. A more detailed discussion of the processing operation 206 and the rendering operation 208 is provided in association with FIG. 6. In a one embodiment, calls to render() methods in individual control objects are performed using a tree traversal sequence. That is, a call to the render() method of a page object results in recursive traversal throughout appropriate server-side control objects in the hierarchy. Alternative methods for calling the render() methods for appropriate control objets may also be employed, including an event signaling or object registration approach. The parentheses designate the "render()" label as indicating a method, as compared to a data value.

In an embodiment of the present invention, the actual creation of the individual server-side control objects may be deferred until the server-side control object is accessed (such as when handling postback input, loading a state, rendering HTML code from the control object, etc.) in operations 206 or 208. If a server-side control object is never accessed for a given request, deferred control object creation optimizes server processing by eliminating an unnecessary object creation operation.

Operation 210 transmits the authoring language data (e.g., HTML code), including the transportable state structure, to the client in an HTTP response. In operation 214, the client receives the HTML code, including the transportable state structure, associated with the web page to be displayed. The transportable state

structure may be stored at the client for return to the server in a subsequent HTTP request. In operation 216, the client system incorporates (e.g., displays) the user interface elements of the new page in accordance with the HTML code received from the HTTP response. It should be understood, however, that incorporation of a user-interface element may include non-display operations, such as providing audio or tactile output, reading and writing to memory, controlling the operation of scripts, etc. In operation 212, the server-side control object hierarchy is terminated. In an embodiment of the present invention, server-side control objects in the hierarchy are created in response to an HTTP request referencing an associated ASP+ resource, and destroyed subsequent to the rendering of authoring language data (e.g., HTML data). In an alternative embodiment, operation 212 may be performed after operation 208 and before operation 210.

FIG. 3 illustrates exemplary modules in a web server used in an embodiment of the present invention. The web server 300 receives an HTTP request 302 into the HTTP pipeline 304. The HTTP pipeline 304 may include various modules, such as modules for logging of web page statistics, user authentication, user authorization, and output caching of web pages. Each incoming HTTP request 302 received by the web server 300 is ultimately processed by a specific instance of an IHTTPHandler class (shown as handler 306). The IHTTP prefix indicates that the class is an Interface of an HTTP handler. The handler 306 resolves the URL request and invokes an appropriate handler factory (e.g., a page factory module 308).

In FIG. 3, a page factory module 308 associated with the ASP+ resource 310 is invoked to handle the instantiation and configuration of the objects declared in the ASP+ resource 310. In one embodiment, an ASP+ resource can be identified or referenced by designating a particular suffix (or file extension such as ".aspx") with a file. When a request for a given ".aspx" resource is first received by the page factory module 308, the page factory module 308 searches the file system for the appropriate file (e.g., the .aspx file 310). The file may contain text (e.g., authoring language data) or data in another format (e.g., bytecode data or encoded data) that may later be interpreted or accessed by the server to service the request. If the physical file exists, the page factory module 308 opens the file and reads the file into memory. If the file cannot be found, the page factory module 308 returns an appropriate "file not found" error message.

After reading the ASP+ resource 310 into memory, the page factory module 308 processes the file content to build a data model of the page (e.g., lists of script blocks, directives, static text regions, hierarchical server-side control objects, server-side control properties, etc.). The data model is used to generate a source listing of a new object class, such as a COM+ ("Component Object Model+") class that extends the page base class. The page base class includes code that defines the structure, properties, and functionality of a basic page object. The source listing is then dynamically compiled into an intermediate language. An intermediate language may include general or custom-built language code, such as COM+ IL code, Java bytecodes, Modula 3 code, SmallTalk code, and Visual Basic code. In an alternative embodiment, the intermediate language operations may be omitted, so that the native instructions are generated directly from the source listing or the source file (e.g., the ASP+ resource 310). A control class library 312 may be accessed by the page factory module 308 to obtain predefined server-side control classes used in the generation of the control object hierarchy.

The page factory module 308 outputs a page object 314, which is a server-side control object that corresponds to the web page 104 of FIG. 1. The page object 314 and its children (i.e., a text box object 318, a button object 320, and another button object 322) comprise an exemplary control object hierarchy 316. Other exemplary control objects are also contemplated in accordance with the present invention, including without limitation objects corresponding to the HTML controls in Table 1, as well as custom control objects. The page object 314 logically corresponds to the web page 104 or FIG. 1. The text box object 318 corresponds to the text box 106 in FIG. 1. Likewise, the button object 320 corresponds to the add button 108 in FIG. 1, and the button object 322 corresponds to the delete button 110 in FIG. 1. The page object 314 is hierarchically related to other control objects on the server. In one embodiment, a page object is a container object that hierarchically contains its children control objects. In an alternative embodiment, other forms of hierarchical relation may be employed, including a dependency relationship. In a more complex control object hierarchy with multiple levels of children, a child object can be a container object for other child objects.

In the illustrated embodiment, the control objects in the control object hierarchy 316 are created and executed on the server 3000, and each server-side

control object "mirrors" a corresponding user interface element on the client. The server-side control objects of this embodiment also cooperate to handle input from the HTTP request 302, to manage the states of server-side control objects, to perform data binding with server-side databases, and to generate authoring language data (e.g., HTML code) used to display a resulting web page at a client. The resulting authoring language data is generated (i.e., rendered) from the server-side control object hierarchy 316 and transmitted to the client in an HTTP response 324. For example, resulting HTML code may embody any valid HTML construct and may reference ACTIVEX-type controls, JAVA applets, scripts, and any other web resources that yield client-side user interface elements (e.g., control buttons, text boxes, etc.) when processed by a browser.

By virtue of declarations made in the ASP+ resource 310, server-side control objects may access one or more non-user-interface server components 330 to provide interaction between the non-user-interface server component 330 and client-side user interface elements. For example, in response to postback input, server-side control objects can raise server-side events to the non-user-interface server components registered for those events. In this manner the non-user-interface server component 330 can interact with the user through user interface elements without programming the code required to display and process these elements.

In summary, an embodiment of the present invention includes server-side control objects that are created and executed on the server to generate HTML code that is sent to a client. The HTML code may, for example, embody any valid HTML constructs and may reference ACTIVEX-type controls, JAVA applets, scripts and any other web resources to produce user interface buttons and other user interface elements at the client. A user at the client may interact with these user interface elements, which logically correspond to the server-side control objects, and send a request back to the server. The server-side control objects are recreated on the server to process the data, events, and other characteristics of the user interface elements so as to generate the next round of HTML code to be transmitted in a response to the client.

With reference to FIG. 4, an exemplary computing system for embodiments of the invention includes a general purpose computing device in the form of a conventional computer system 400, including a processor unit 402, a system

memory 404, and a system bus 406 that couples various system components including the system memory 404 to the processor unit 400. The system bus 406 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 408 and random access memory (RAM) 410. A basic input/output system 412 (BIOS), which contains basic routines that help transfer information between elements within the computer system 400, is stored in ROM 408.

The computer system 400 further includes a hard disk drive 412 for reading from and writing to a hard disk, a magnetic disk drive 414 for reading from or writing to a removable magnetic disk 416, and an optical disk drive 418 for reading from or writing to a removable optical disk 419 such as a CD ROM, DVD, or other optical media. The hard disk drive 412, magnetic disk drive 414, and optical disk drive 418 are connected to the system bus 406 by a hard disk drive interface 420, a magnetic disk drive interface 422, and an optical drive interface 424, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, programs, and other data for the computer system 400.

Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 416, and a removable optical disk 419, other types of computer-readable media capable of storing data can be used in the exemplary system. Examples of these other types of computer-readable mediums that can be used in the exemplary operating environment include magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), and read only memories (ROMs).

A number of program modules may be stored on the hard disk, magnetic disk 416, optical disk 419, ROM 408 or RAM 410, including an operating system 426, one or more application programs 428, other program modules 430, and program data 432. A user may enter commands and information into the computer system 400 through input devices such as a keyboard 434 and mouse 436 or other pointing device. Examples of other input devices may include a microphone, joystick, game pad, satellite dish, and scanner. These and other input devices are often connected to the processing unit 402 through a serial port interface 440 that is coupled to the

system bus 406. Nevertheless, these input devices also may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB). A monitor 442 or other type of display device is also connected to the system bus 406 via an interface, such as a video adapter 444. In addition to the monitor 442, computer systems typically include other peripheral output devices (not shown), such as speakers and printers.

The computer system 400 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 446. The remote computer 446 may be a computer system, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer system 400. The network connections include a local area network (LAN) 448 and a wide area network (WAN) 450. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, the computer system 400 is connected to the local network 448 through a network interface or adapter 452. When used in a WAN networking environment, the computer system 400 typically includes a modem 454 or other means for establishing communications over the wide area network 450, such as the Internet. The modem 454, which may be internal or external, is connected to the system bus 406 via the serial port interface 440. In a networked environment, program modules depicted relative to the computer system 400, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary, and other means of establishing a communication link between the computers may be used.

In an embodiment of the present invention, the computer 400 represents a web server, wherein the cpu 402 executes a page factory module on an ASP+ resource stored on at least one of storage media 416, 412, 414, 418, 419, or memory 404. HTTP responses and requests are communicated over the LAN 448 that is coupled to a client computer 446.

FIG. 5 illustrates a process flow diagram representing server-side processing of a page object and other control objects in an embodiment of the present invention. In operation 500, a page object constructor is called by the page factory module 308

(see FIG. 3). As a result, a page object (see e.g., the page object 314 in FIG. 3) is created to "mirror" the web page user interface element on the client. In operation 502, the page factory module calls the ProcessRequest member function of the page object, which initiates the staged operations for processing the HTTP request received from a client. In a first stage of one embodiment of the present invention, a server-side Create Operation (not shown) may create the descendant server-side control objects contained in the control object hierarchy of the page object, that is, constructors for child control objects are recursively called to create the control objects during the lifetime of the processing of the HTTP request processing.

In an alternate embodiment, however, creation of child control objects is deferred until the control object is required for a given processing step (e.g., handling a postback event, handling postback data, loading or saving a viewstate, resolving data binding, or rendering HTML code for the corresponding user interface element). The latter embodiment, which is said to implement, "deferred control object creation", is an optimization that can alleviate unnecessary CPU and memory utilization. For example, a user input event received from the client may result in the creation of a completely different web page. In this case, it is unnecessary to instantiate an entire control object hierarchy of the previous page only to process an event that immediately results in the termination of the control object hierarchy and the instantiation of a new and different control object hierarchy for a new page.

In response to the server call to the page object's ProcessRequest method, operations 504 through 520 may be executed by the page object and by individual descendant control objects, depending in part on the data of a given HTTP request. In an embodiment of the present invention, the operations 504-520 are performed for each individual object in the order illustrated in FIG. 5; however, a given operation for one object may occur out of order or not at all with respect to a given operation of another object, depending on the HTTP request. For example, a first object may perform its Init operation 504 and its Load operation 506, and begin postback data processing operation 508, before a descendant control object performs its own Init operation 504 and Load operation 506 by virtue of deferred control object creation. The order of operation processing by the page object and the descendent control objects depends on various factors, including without limitation the nature of the

data in the HTTP request, the configuration of the control object hierarchy, the current state of the control objects, and whether deferred control object creation is implemented.

The Init operation 504 initializes a control object after it is created by executing any server-side code associated with initialization in the dynamic content file. In this manner, each server-side control object may be customized with specific server-side functionality that is declared in the dynamic content file. In an embodiment of the present invention, dynamic content code intended to customize or extend the base page control classes as declared by the page developer in the ASP+ resource on the server. When the ASP+ resource is compiled, the declared code is included in the appropriate initialization code (e.g., the Init() methods of the page object and the descendent control objects). The Init operation 504 executes this code to customize or extend the page base class and the base classes for descendent control objects.

In an embodiment of the present invention, state management of the server-side control objects is supported in a Load operation 506 and a Save operation 516, which use a transportable state structure to accommodate the stateless model for client server systems by restoring server-side control objects to their previous states. In one embodiment, the state is communicated to and from the server in one or more hidden HTML fields of an HTTP request/response pair, although other transportable state structures are contemplated within the scope of the present invention, including cookies and visible fields.

In a given sequence of requests and responses relating to the current page between a client and a server, the states of one or more control objects are recorded into a transportable state structure by the Save operation 516 after the processing of a previous request. In an embodiment of the present invention, additional state information is also included in the transportable state structure, including hierarchical information or control object identifiers to allow the server to associate a given state with the appropriate control object. In a subsequent HTTP request, the state information is returned to the server in the transportable state structure. The server extracts the state information from the received transportable state structure and loads the state data into the appropriate control objects within the control object hierarchy to restore each control object to its state as it existed prior to a previous

HTTP response. After the current request is processed, the states of one or more server-side control objects are again recorded into the transportable state structure by the Save operation 516, and the transportable state structure is returned to the client in the next HTTP response.

As a result of the Load operation 506, each server-side control object is placed in a state consistent with its state prior to a previous HTTP response. For example, if a text box control object includes a property value equaling "JDoe" prior to a previous HTTP response, the Load operation 506 restores the same control object to its previous state, in part by loading the text string "JDoe" into the property value. In addition, whether the state of a given object is stored and restored is configurable.

In summary of one embodiment of the present invention, the state of one or more server-side control objects is "saved" after processing. The saved state information is transmitted to the client in a response. The client returns the saved state information to the server in a subsequent response. The server loads the state information a freshly instantiated server-side control object hierarchy, such that the state of the hierarchy is restored to its previous state.

An alternative embodiment may maintain the state information on the server or at some other web location accessible by the server during the round trip from the server to the client, and then back to the server. After the client request is received by the server, this state information may be retrieved by the server and loaded into the appropriate server-side control object(s) in the control object hierarchy.

In operation 508, postback data received from the HTTP request is processed. Postback data may be included in the payload of the HTTP request in key-value pairs, in a hierarchical representation (e.g., XML), or in other data representations, such as RDF ("Resource Description Framework"). Operation 508 parses the payload to identify a unique identifier of a server-side control object. If the identifier (e.g. "page1:text1") is found and the identified server-side control object exists in the control object hierarchy, the corresponding postback data is passed to the control object. For example, referring to FIG. 1, a unique identifier associated with textbox 106 and the text "JDoe" are communicated in the payload of the HTTP request 114 to the web server 116. Operation 508 parses the payload of the HTTP request 114 and obtains the unique identifier of the textbox 106 and its

associated value (i.e., "JDoe"). Operation 508 then resolves the unique identifier of the textbox 106 to identify the corresponding server-side control object and passes the "JDoe" value to the object for processing.

As discussed with regard to the Load operation 506, the property values of server-side control objects may be restored to their previous states. In response to the receipt of postback data, the server-side control object determines whether the passed-in postback value causes a change from the corresponding property's previous value. If so, the change is logged in a change list to indicate a data change for the associated control object. After all postback data has been processed within the control object hierarchy, a call may be made to a control object method to raise one or more postback data changed events to one or more non-user-interface server components, such as a stock price look-up application running on the server. An example of a postback data changed event is an event indicating that postback data has caused a property of a server-side control object to change. In an exemplary embodiment, such an event can be sent to a system-provided event queue so that a server component that is registered to process the event may be invoked. In this manner, a server-side non-user-interface server component can respond to events triggered by a change in data of a server-side control object. Alternative methods of implementing events are also contemplated in the scope of the present invention, including using application-provided event queues, polling, and processing interrupts.

In operation 510, postback events are handled. Postback events are communicated in the payload of the HTTP request. Operation 510 parses a specified event target (e.g., labeled "__EVENTTARGET" in an embodiment of the present invention) identifying the server-side control object to which the event is directed. Furthermore, operation 510 parses the located event arguments, if any, and provides the event argument (e.g., labeled "__EVENTARGUMENT" in an embodiment of the present invention) to the specified server-side control object. The control object raises its events for processing by the non-user-interface server component (e.g., a server-side stock price look-up applications) associated with the dynamic content file.

Operation 512 resolves data binding between the server-side control objects and one or more databases accessible by the server. In an embodiment of the present

invention, properties of server-side control objects may be associated (or data bound) to properties of a parent data binding container, such as a table in a server-side application database. During the data binding operation 612, the page framework may update a data bound control object property with the value of the corresponding parent data binding container property. In this manner, user interface elements on the web page of the next response accurately reflect updated property values, because the control object properties to which the user interface elements correspond have been automatically updated during the data binding operation 512. Likewise, control object properties can also be updated to the parent data binding container fields, thereby updating a server-side application database with postback input from a server-side control object.

Operation 514 performs miscellaneous update operations that may be executed before the control object state is saved and the output is rendered. Operation 516 requests state information (i.e., viewstate) from one or more control objects in the control object hierarchy and stores the state information for insertion into a transportable state structure that is communicated to the client in the HTTP response payload. For example, a "grid" control object may save a current index page of a list of values so that the "grid" control object may be restored to this state after a subsequent HTTP request (i.e., in operation 506). As described above, the view state information represents the state of the control object hierarchy prior to any subsequent actions by the client (e.g., before the HTTP response is sent to the client). When the view state information is returned, it will be used to place the control object hierarchy in that previous state prior to processing any client postback input or databinding.

The render operation 518 generates the appropriate authoring language output (e.g., HTML data) for communication to the client in an HTTP response. Rendering is accomplished through a top-down hierarchical tree walk of all server-side control objects and embedded rendering code. Operation 520 performs any final cleanup work (e.g., closing files or database connections) before the control object hierarchy is terminated. Processing then returns to operation 502 and proceeds to operation 522 where the page object is terminated by calling its destructor.

FIG. 6 illustrates an exemplary portion of a dynamic content file (e.g., an ASP+ resource) in an embodiment of the present invention. Line 1 of the ASP+ resource 600 is a start tag of an HTML file and is declared in the ASP+ resource 600 as a literal. A literal corresponds to a server-side literal control object in the server-side control object hierarchy. The literal control object corresponding to line 1 is given an index of "0" because it is the first control object resulting from a declaration in the ASP+ resource 600. At render time, the literal control object merely generates the "<html>" text and a new line for inclusion in the HTTP response. Lines 2 through 12 of the ASP+ resource 600 represent a code declaration block, which is executed on the server (i.e., as indicated by the "runat = server" attribute on line 2). In the exemplary ASP+ resource 600, the code declaration block does not result in the instantiation of a server-side control object in the control object hierarchy. Instead, the code declaration block results in server-side code being "wired" to or associated with the server-side control object declared in line 15. Line 13 of the ASP+ resource 600 is a start tag of the body of the HTML file and is declared in the ASP+ resource 600 as a literal to result in a literal control object having an index equaling "1".

The declaration on line 14 declares a server-side form control object (index= "2") that is to be instantiated into the control object hierarchy. The tags on line 17 and 18 are closing tags to be represented by a server-side literal control object (index="3") in the control object hierarchy. The closing tag on line 16 closes the <form> declarations corresponding to the form control object.

In a subsequent level of hierarchy, the declaration on line 15 declares a server-side label control object (index="1") using the HTML control tag "span" and having the identifier attribute ("id") equaling "Message". Indices "0" and "2" of this level of hierarchy are allocated respectively to a literal control object (index = "0") for a preceding white space literal (e.g., tabs, new lines, spaces, etc.) and a literal control object (index = "2") of a subsequent white space literal.

FIG. 7 illustrates resultant code generated by one or more server side control objects in response to the exemplary portion of the dynamic content file of FIG. 6. Only the form, state management, and span sections are illustrated in an HTML code portion 700 to facilitate this discussion. Other tags, such as the <html> and <body> tags, are not included in the HTML code portion 700, although the HTML code for

these tags would be included in a set of complete HTML code resulting from the ASP+ resource of FIG. 6.

Lines 1 and 5 are starting and ending tags of the form section of the HTML code and are generated by the server-side form control objects declared at lines 14 through 16 of FIG. 6. Line 4 includes the HTML code for a label displaying the date of the Last Post Back operation (i.e., "Last Post Back: 5/6/99"), which was declared in line 15 of FIG. 6.

The hidden fields on lines 2 and 3 represent an embodiment of a transportable state structure in accordance with the present invention. On line 2, the state information of the server-side control objects declared in the ASP+ resource 600 is recorded in the hidden field named "__VIEWSTATE". The value of the _VIEWSTATE field is a text string representing the state values, property types, and hierarchical information of the control objects in the control object hierarchy. Hierarchical information is also included in the __VIEWSTATE field to allow the Load operation to traverse the hierarchy and load a given state value into a property of an appropriate control object in the hierarchy.

In an embodiment of the present invention, a client interacts with a web server using a sequence of HTTP request/response pairs. Between a response to the client and the next request received by the server, the server may not maintain the state of the server-side control objects associated with a given ASP+ resource or a given connection with the client. Instead, the state information for a given control object hierarchy is sent to the client in a transportable state structure (e.g., in the HTTP response), and returned to the server in the transportable state structure (e.g., in the next HTTP request).

Line 3 represents a hidden field named "__VIEWSTATEMAC" having a value of "434333433". The __VIEWSTATEMAC is an integrity code in the transportable state structure that is used by the Load operation to verify that the _VIEWSTATE value was not corrupted at the client. The __VIEWSTATEMAC value is initially calculated at the server from the contents of the __VIEWSTATE value and communicated to the client in a response. When the client returns the transportable state structure, the web server calculates a new integrity code from the contents of the received __VIEWSTATE value. If the received integrity code and the newly calculated code are equal, then the web server assumes that the received

_VIEWSTATE value is valid or otherwise uncorrupted (i.e., is the same as the _VIEWSTATE value that was previously sent to the client). In an embodiment of the present invention, an MD5 algorithm is used to calculate the integrity code. MD5 is an algorithm created in 1991 by Professor Ronald Rivest for use in creating digital signatures. MD5 is a one-way hash function, meaning that it takes a message and converts it into a fixed string of digits, also called a message digest. Such methods may include the use of a secret key to discourage tampering by unauthorized individuals or programs. Other embodiments, however, may include alternative integrity coding techniques, including generating a code based on the MD4 algorithm or any other hash algorithm, or the length of the __VIEWSTATE field.

In an embodiment of the present invention, the structure of the _VIEWSTATE field of line 2 represents a hierarchical nesting of control objects and their states (e.g., state values and their associated property data types). The tags used in an exemplary __VIEWSTATE field are described in Table 1. Tags that include a "/" character represent closing tags.

Tag	Description
<s>, </s>	string value
<a*x*>, 	array having *x* elements
<i>	integer value
<h*x*>, </H>	hash table having *x* table entries
<K>, </K>	hash table key
<V>, </V>	hash table value
,	Boolean value
<d>, </d>	date/time value
<c>, </c>	currency value
<A>, </A>	array list
<n>, </n>	null value

TABLE 1

It should be understood that an exemplary encoding embodiment is disclosed herein, although in alternative embodiments, other encoding methods may be employed. For example, hierarchy information, state values and their associated

property data types may be designated using unique hierarchical identifiers for each server-side control object, or an XML-related data format may be employed to represent the data in a transportable state structure. In addition, known encryption and compression techniques may be use to provide security and to reduce the size of the transportable state structure.

FIG. 8 illustrates the control object hierarchy corresponding to the ASP+ resource of FIG. 6 and the __VIEWSTATE field value of FIG. 7 in an embodiment of the present invention. A file check block 802, while not a control object component of the control object hierarchy, is illustrated as a top level of hierarchy based on the __VIEWSTATE field structure, such as shown in FIG. 7. A more detailed description of the file check block 802 is provided with regard to FIG. 9. Alternatively, omitting the file check block 802 or merging it into the level of hierarchy corresponding to a page control object 804 is contemplated within the scope of the present invention.

The page control object 804 corresponds to a special control object instantiated as the top level of the control object hierarchy and corresponding to the resulting web page itself. In an embodiment of the present invention, no specific declaration (i.e., a declaration having the "runat = server" attribute) is required to cause an instantiation of the page object 804.

At the next level of hierarchy, control objects 806, 808, 810, and 812 are instantiated as child control objects contained by or hierarchically related to the page object 804. Each control object in a level of hierarchy is assigned a zero-based index based on its top-down order in the ASP+ resource. For example, literal control object 806 corresponds to the "<html>" text and associated white spaces on line 1 of the ASP+ resource 600 in FIG. 6. Accordingly, the literal control object 806 is assigned an index equaling "0". In the illustrated embodiment of the present invention, the code declaration block extending from lines 2-12 does not result in the instantiation of a corresponding control object. The literal control object 808 corresponds to the "<body>" text and associated white space on line 13 of the ASP+ resource 600 of FIG. 6. Accordingly, the literal control object 808 is assigned an index equaling "1". The form control object 810 corresponds to the form declaration on lines 14 through 16 of the ASP+ resource 600 of FIG. 6. Accordingly, the form control object 810 is assigned an index equaling "2". The

literal control object 812 corresponds to the closing tags and associated white space on lines 17 and 18 of the ASP+ resource 600 of FIG. 6. Accordingly, the literal control object 812 is assigned an index equaling "3".

At the next level of hierarchy, control objects 814, 816 and 818 are instantiated as child control objects contained by or hierarchically related to the form control object 810. The literal control object 814 corresponds to white space preceding the span declaration on line 15 of the ASP+ resource 600 of FIG. 6. Accordingly, the literal control object 814 is assigned an index of "0", because it is the first literal text (albeit white space) encountered in association with this level of hierarchy in the ASP+ resource. Likewise, the literal control object 818 corresponds to white space following the span declaration of line 15 in the ASP+ resource 600 of FIG. 6. Accordingly, the literal control object 818 is assigned an index of "2". The label control object 816 corresponds to the declaration on line 15 of the ASP+ resource 600 in FIG. 6. Accordingly, the label control object 816 is assigned an index equaling "1".

FIG. 9 illustrates a nested version of the __VIEWSTATE field value of FIG. 7. The nested version is depicted herein to facilitate discussion of the hierarchical nature of the __VIEWSTATE field. Line 1 of the nested version 900 corresponds to the beginning of the nested file check block 802 of FIG. 8. Line 1 specifies a two-element array, comprising a string on line 2 as the first element and a two-element array of line 3 as the second element. Line 2 specifies a string value representing a file check code. In an embodiment to the present invention, the file check code is calculated as a hash value of the associated ASP+ resource (e.g., ASP+ resource 600 of FIG. 6). The array started on line 3 represents the beginning of state information for the next level of hierarchy, namely the page control object 804 of FIG. 8. The null value on line 4 represents the first element of the two-element array of line 3, which is capable of holding state information relating to properties of the page object itself. In the illustrated embodiment, no state information is recorded for the page object, so the value equals "null".

As shown in line 5, the second element of the two-element array of line 3 indicates a hash table including state information for the page control object's children (namely, control objects 806, 808, 810 and 812 of FIG. 8). The hash table of line 5 includes one hash table entry, as indicated by the "1" in the tag <H1>. The

<K>, </K> tags on lines 6 and 8 encompass the key for the hash table entry, namely the integer value "2" indicated in line 7, which corresponds to the index of the form control object 810 of FIG. 8. The _VIEWSTATE value depicted in FIG. 9 includes no state information for the literal control objects 806, 808, and 812 of FIG. 8. Alternatively, if other control objects having saved state information or child control objects existed at the same level of hierarchy as the form control object 810, then the hash table at this level may have more than one hash table entry to accommodate the additional control objects at this level, and the key to each additional hash table entry would be the index of the corresponding control object.

The value of the hash table entry is encompassed by the <V>, </V> tags of lines 9 and 24 and includes another two-element array to define the state of the form control object and its children. The first element of the array specifies the saved state information for the form control object (i.e., represented by a null value in this embodiment).

The second element specifies another one-entry hash table representing the state information for the children of the form control object. The <K>, </K> tags on lines 13 and 15 encompass the key for the single hash table entry, namely the integer value "1" indicated in line 14, which corresponds to the index of the label control object 816 of FIG. 8. The _VIEWSTATE value depicted in FIG. 9 includes no state information for the literal control objects 814, and 818 of FIG. 8.

The value of the hash table entry is encompassed by the <V>, </V> tags of lines 16 and 21 and includes another two-element array to define the state of the label control object and its children. The first element of the array specifies the saved state information for a property of the label control object, which is the string value "InnerHtml=5/16/99" specified in line 18. The property name, "InnerHtml" equals the state value "5/16/99" and is of type "string". The semicolon represents the end of the state value for a given property. If the label control object of line 18 included additional properties for which the state was saved, each state data value and its associated property would be delimited by an ending semicolon. The null value indicated in line 19 as in the second element of the array indicates that label control object does not contain any children. The lines 20-27 represent closing tags for preceding starting tags in FIG. 9.

In an alternative embodiment, a serialization format called Limited Object Serialization (LOS) format is used in a transportable state structure. In general, LOS format specifies a hash table of name/value pairs for each control object, wherein each hash entry contains either state information for a property of the control object or a nested hash table of a child control object.

Table 2 illustrates an exemplary grammar of the LOS format.

control object	• value type-table$_{opt}$ name-table$_{opt}$	• h<name1;value1>\t50System.Drawing.Color\n1BackColor
value	• typed-value • untyped-value • typed-array-value • untyped-array-value • untyped-hashtable-value	• 50<red> • <red> • a50<red;blue;green> • a<red;blue;green> • h<name1;value1>
typed-value	• type-ref value-list-start value-ref value-list-end	• 50<red>
typed-array-value	• array-modifier type-ref value-list-start array-value-ref value-list-end	• a50<red;blue;green>
untyped-value	• value-ref	• <red>
untyped-array-value	• array-modifier value-list-start array-value-ref value-list-end	• a<red;blue;green>
value-list-start	• <	
value-list-end	• >	
value-list-separator	• ;	
array-modifier	• a	
hashtable-modifier	• h	
untyped-hashtable-value	• hashtable-modifier value-list-start hashtable-value-ref value-list-end	• h<name1;value1>
value-ref	• string-value • bin-ref base64-persisted-object	• \"This is a string value.\"

bin-ref	• escape-char b	• \b
array-value-ref	• $value_1$;[$value_2$;[$value_n$]]	• red;blue;green
hashtable-value-ref	• $name\text{-}ref_1$;$value_1$ [value-list-separator $name\text{-}ref_2$;$value_2$ [value-list-separator $name\text{-}ref_n$;$value_n$]]	• <name1;value1; name2;value2; name3;value3>
name-ref	• string-name-number • string-name	• 1 • BackColor
type-ref	• known-type-number • string-type-number • string-type	• 10 • 50 • System.Drawing.Color
type-table	• type-table-start $string\text{-}type\text{-}number_1$ $string\text{-}type_1$[value-list-separator $string\text{-}type\text{-}number_2$ $string\text{-}type_2$[value-list-separator $string\text{-}type\text{-}number_n$ $string\text{-}type_n$;]]	• \t50System.Drawing.Color;51System.Drawing.Font
name-table	• name-table-start $string\text{-}number_1$ $string\text{-}name_1$[value-list-separator $string\text{-}name\text{-}number_2$ $string\text{-}name_2$[value-list-separator $string\text{-}name\text{-}number_n$ $string\text{-}name_n$;]]	• \n1BackColor;2ForeColor
known-type-number	• 0 1 2 3 4 5 6 7 8 9 • 10 11 12 13 14 15 16 17 18 19 • 20 21 22 23 24 25 26 27 28 29 • 30 31 32 33 34 35 36 37 38 39 • 40 41 42 43 44 45 46 47 48 49	
type-table-start	• escape-char t	• \t
name-table-start	• escape-char n	• \n
string-terminator-char	• value-list-separator value-list-end ' ' " " \	

escaped-string	• escape-char string-terminator-char	• \"
escape-char	• \	
string-number	• 0 1 2 3 4 5 6 7 8 9	
base64-persisted-object	• Any object that is BinarySerialized and then base64 encoded	
string-name	• Any string that is set off by a string delimiter	
string-type	• Any string that matches a COM+ class name and that is set off by a string delimiter	
string-value	• Any string that is set off by a string delimiter that can be converted using the type's TypeConverter	

Table 2

As a first example of how the LOS format may be used, consider the entry:

h<name1;value1;name2;value with \; escaped \> \"characters\">

which defines a hash table of 2 name/value pairs, wherein the values are all strings. The first value is named "name1" and equals the string "value1". The second value is named "name2" and equals the string "value with ; escaped > "characters"".

As a second example of how LOS format may be used, consider the entry:

h<control1;h<1;50<blue>;text;hello>control2;h<1;50<red>;
control3;>\t50System.Drawing.Color\n1BackColor

which defines a hash table having three hash entries: "control1", "control2", and "control3". The first hash entry is named "control1" and includes a child hash table having two entries, which are name/value pairs. The name of the first entry in the

control1 hash table uses the index "1" to reference the name "BackColor", which is defined at the end of the example. The value of the first entry equals "blue" and is of type "System.Drawing.Color", which is specified by the index "50" and defined near the end of the example. The name/value pair of the second entry in the control1 hash table includes the name "text" and the value "hello". The "control2" hash entry includes child hash table having one hash entry, which also uses the index "1" to reference the name "BackColor". The value equals "red" and is of type "System.Drawing.Color". The "control3" hash table is empty. In a manner similar to that of the previously disclosed embodiment, hierarchy is specified by the nesting of the hash tables; however, other methods of describing hierarchy are contemplated within the scope of the present invention, including the use of hierarchical identifiers.

FIG. 10 illustrates a process flow diagram for receiving a transportable state structure and loading state information stored therein into control objects of a control object hierarchy in an embodiment of the present invention. In the illustrated embodiment, a transportable state structure may not be sent from the server (and, therefore, not returned to the server by the client) unless at least one property changed for at least one server-side control object. As such, the illustrated flow diagram assumes that at least one element of state information exists in the received transportable state structure. In an alternative embodiment, the transportable state structure may be round-tripped between the server and the client, even if no state information exists therein. In such embodiment, a decision operation (not shown) could operate to abort the state management process if no state information exists in the transportable state structure.

Receiving operation 1000 receives a transportable state structure from a client, such as in an HTTP request. Marking operation 1002 sets an initial state indication for all control objects in the server-side hierarchy. Reading operation 1004 reads the received transportable state structure to extract the state information and corresponding property types.

Checking operation 1006 performs an integrity check on the transportable state structure (e.g., using the __VIEWSTATEMAC Value). In an embodiment of the present invention, checking operation 1006 involves (1) reading a received integrity code from the transportable state structure; (2) calculating an integrity code

of its own from at least the state information included in the transportable state structure; and (3) comparing the received integrity code with the newly calculated integrity code to determine if the transportable state structure has been corrupted during the round trip to the client. If the transportable state structure is determined to be corrupted, the server-side code can respond to handle the exception, including by aborting the load operation, continuing with processing in the absence of state information or raising an error.

A verification operation 1008 verifies the file check code included in the transportable state structure. In an embodiment of the present invention, verification operation 1008 includes (1) reading the received file check code from the __VIEWSTATE field; (2) calculating its own file check code from at least the contents of the ASP+ resource store on the server; (3) and comparing the received file check code with the newly calculated from file check code to determine whether the state information in the transportable state structure corresponds to the same version of the ASP+ resource on the server. Verification operation 1008 is used to verify that the ASP+ resource on the server did not change during the transportable state structure's round trip to the client. If the ASP+ resource on the server did change, then the server may discard the received state information, abort the load operation and/or signal an error to the client. In one embodiment, the control object hierarchy is created in its initial state and the load operation is aborted before the server proceeds with processing the request. Traversing operation 1010 traverses to the page control object of the server-side control object hierarchy.

Parsing operation 1012 parses state information for a control object hierarchy corresponding to a control object in the nested __VIEWSTATE field. In the first iteration, the corresponding control object is the page control object traversed to in traversing operation 1010. In subsequent iterations, the corresponding control object is a descendant control object of the page control object and is located by the traversal operation 1028. . The parsing operation 1012 may extract a state value and convert the value, which is initially in a string format when it is parsed, into the given property type. Loading operation 1014 loads the state information parsed from the __VIEWSTATE value into a property of a control object. Indicating operation 1016 sets a change state indication for the current control object. This indication may be later used to only save the state of those control objects in which

the state has changed from an initial state, thereby minimizing the size of the transportable state structure. Alternatively, all state information may be saved. During postback data handling, postback event handling, and data binding operations in the control object hierarchy, changes made to the state of a control object may also result in the setting of the changed state indication for individual control objects. In an embodiment of the present invention, however, any data change indication that is set due to data binding operations is reset so that the state of that property is not recorded in the transportable state structure, thereby further minimizing the size of the transportable state structure. The state of such a property is dependent on the data specified by the binding relationship, which will be updated in the data binding operation 512 of FIG. 5.

Decision operation 1018 directs processing to operation 1022, if additional state information exists for the current control object. If so, operation 1022 parses the state information for the next control object property in the current control object and directs processing to operation 1014. If no additional state information is available for the current control object, decision operation 1018 directs processing to decision operation 1020, which directs processing to processing operation 1024 if no other control object state information is available in the __VIEWSTATE value. If decision operation 1020 determines that additional control objects exist for which states have not been processed, processing proceeds to operation 1026, which parses the state information for the next control object from the __VIEWSTATE value. Traversing operation 1028 traverses to the next control object corresponding to the new state information parsed in operation 1026. Thereafter, processing proceeds to loading operation 1014. The recursive parsing operations of FIG. 10 implement a deserializing operation in an embodiment of the presenting invention.

FIG. 11 illustrates a process flow diagram for saving state information from control objects of a control object hierarchy into a transportable state structure for transmission to the client in an embodiment of the present invention. Initializing operation 1100 initializes a transportable state structure. Calculating operation 1102 calculates a file check code based on the current version of the ASP+ resource. In an embodiment of the present invention, the file check code is based on a hash algorithm (e.g., MD5 or other) of the ASP+ resource contents, such as merely hashing the file's contents or some specified component of the file. Loading

operation 1104 loads the calculated file check code into the transportable state structure. Traversing operation 1106 traverses to the page object in the control object hierarchy.

In an embodiment of the present invention, a server-side control object can be created with a "MaintainState" property, which indicates whether the property values of the control object (and its children) should be saved into the transportable state structure. If not, the traversal operation 1106 may skip the server-side control object and its children in this process. Otherwise, if the state of the control object is to be maintained in accordance with the "MaintainState" property, then the traversal operation 1106 will traverse into the control object.

Decision operation 1108 directs processing to reading operation 1115, which reads the new state value and its type from the control object property. Operation 1110 records the new state value and its type before proceeding to operation 1111. For the purposes of this discussion, if no property exists within the current control object, the property is represented by a "null" value in the _VIEWSTATE field. If the current control object's property is still in its initial state in operation 1108, processing proceeds to decision operation 1111. If another property exists within the current control object in operation 1111, processing proceeds to decision operation 1108.

If no other property exists in the current control object in operation 1111, processing proceeds to decision operation 1112, which directs processing to traversal operation 1114 if another control object exists in the hierarchy. Traversal operation 1114 traverses to the next control object in the control object hierarchy and proceeds to decision operation 1108 to access the control object's property. If no other control object exists in the hierarchy in operation 1112, processing proceeds to serializing operation 1116 which serializes the new state information (including state value types) obtained in the previous operations and stores them into the transportable state structure. Operation 1118 calculates an integrity code (such as by using an MD5 algorithm), and operation 1120 stores the integrity code into the transportable state structure, such as in the _VIEWSTATEMAC field. It should be understood that the integrity code operations and the filed check code operations are optional and may be omitted in an alternative embodiment of the present invention. Operation 1122 transmits the transportable state structure to the client, such as in an

HTTP response containing the rendered HTML code generated by the control object hierarchy.

The embodiments of the invention described herein are implemented as logical steps in one or more computer systems. The logical operations of the present invention are implemented (1) as a sequence of processor-implemented steps executing in one or more computer systems and (2) as interconnected machine modules within one or more computer systems. The implementation is a matter of choice, dependent on the performance requirements of the computer system implementing the invention. Accordingly, the logical operations making up the embodiments of the invention described herein are referred to variously as operations, steps, objects, or modules.

The above specification, examples and data provide a complete description of the structure and use of embodiment of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

4 Brief Description of Drawings

FIG. 1 illustrates a web server for dynamically generating web page content for display on a client in an embodiment of the present invention.

FIG. 2 illustrates a flow diagram of operations for processing and rendering client-side user interface elements using server-side control objects in an embodiment of the present invention.

FIG. 3 illustrates exemplary modules in a web server used in an embodiment of the present invention.

FIG. 4 illustrates an exemplary system useful for implementing an embodiment of the present invention.

FIG. 5 illustrates a process flow diagram representing processing of a page object in an embodiment of the present invention.

FIG. 6 illustrates an exemplary portion of a dynamic content file (e.g., an ASP+ resource) in an embodiment of the present invention.

FIG. 7 illustrates resultant code generated by one or more server side control objects in response to the exemplary portion of the dynamic content file of FIG. 6.

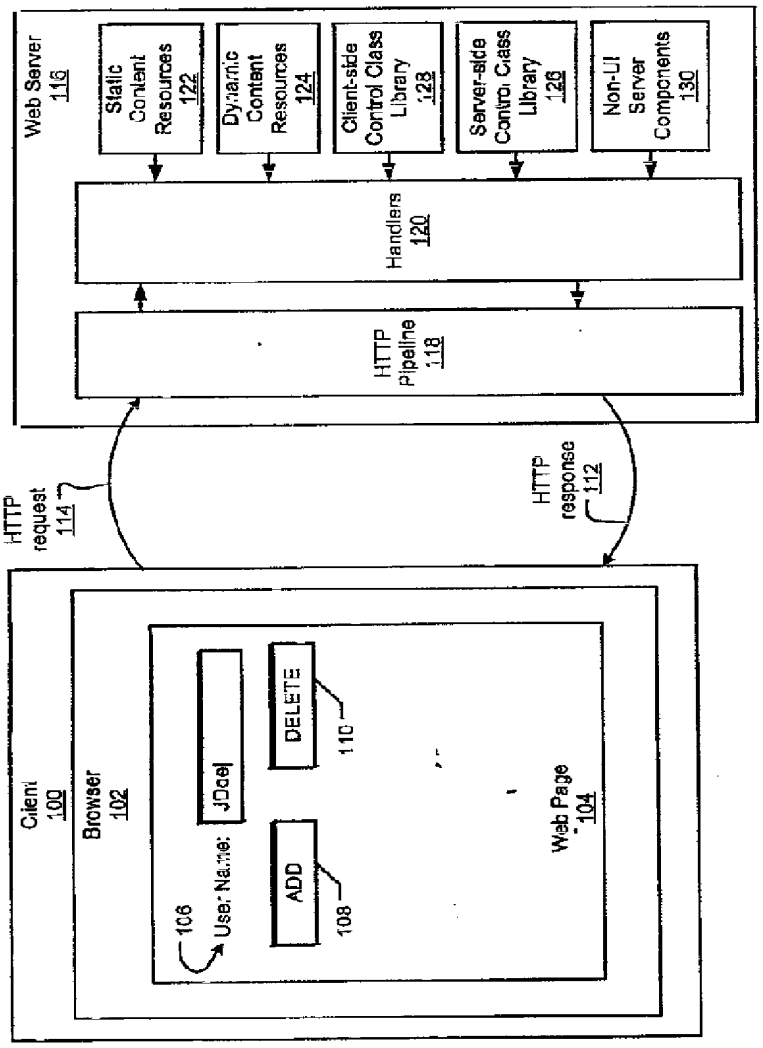
FIG. 8 illustrates the control object hierarchy corresponding to the ASP+ resource of FIG. 6 and the __VIEWSTATE field value of FIG. 7.

FIG. 9 illustrates a nested version of the __VIEWSTATE field value of FIG. 7.

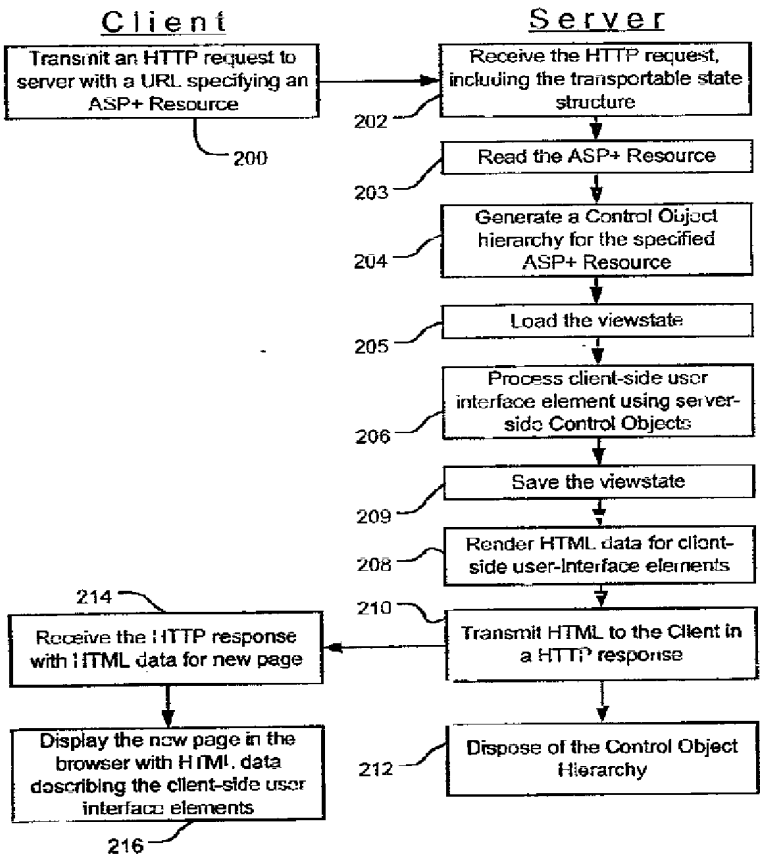
FIG. 10 illustrates a process flow diagram for receiving a transportable state structure and loading state information stored therein into control objects of a control object hierarchy in an embodiment of the present invention.

FIG. 11 illustrates a process flow diagram for saving state information from control objects of a control object hierarchy into a transportable state structure for transmission to the client in an embodiment of the present invention.

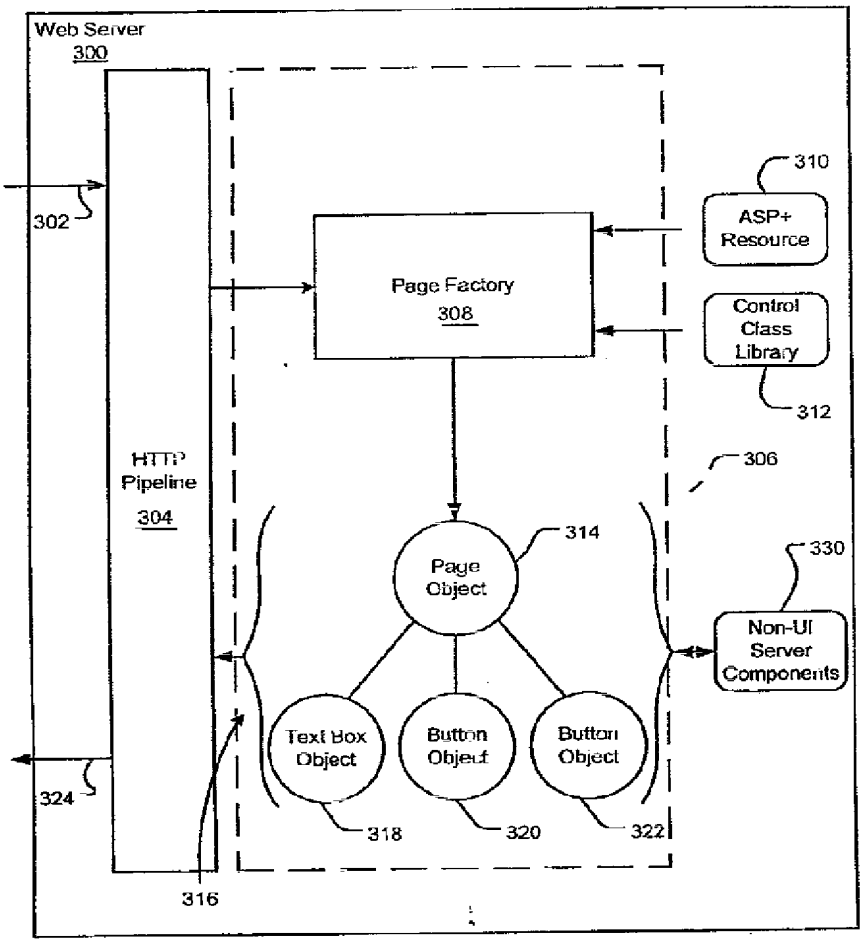
【図1】


Web Server
116
Static Content Resources
122
Dynamic Content Resources
124
Client-side Control Class Library
123
Server-side Control Class Library
126
Non-UI Server Components
130
Handlers
120
HTTP Pipeline
118
HTTP request
114
HTTP response
112
Client
100
Browser
102
JDoe
DELETE
110
ADD
108
106
User Name:
Web Page
104

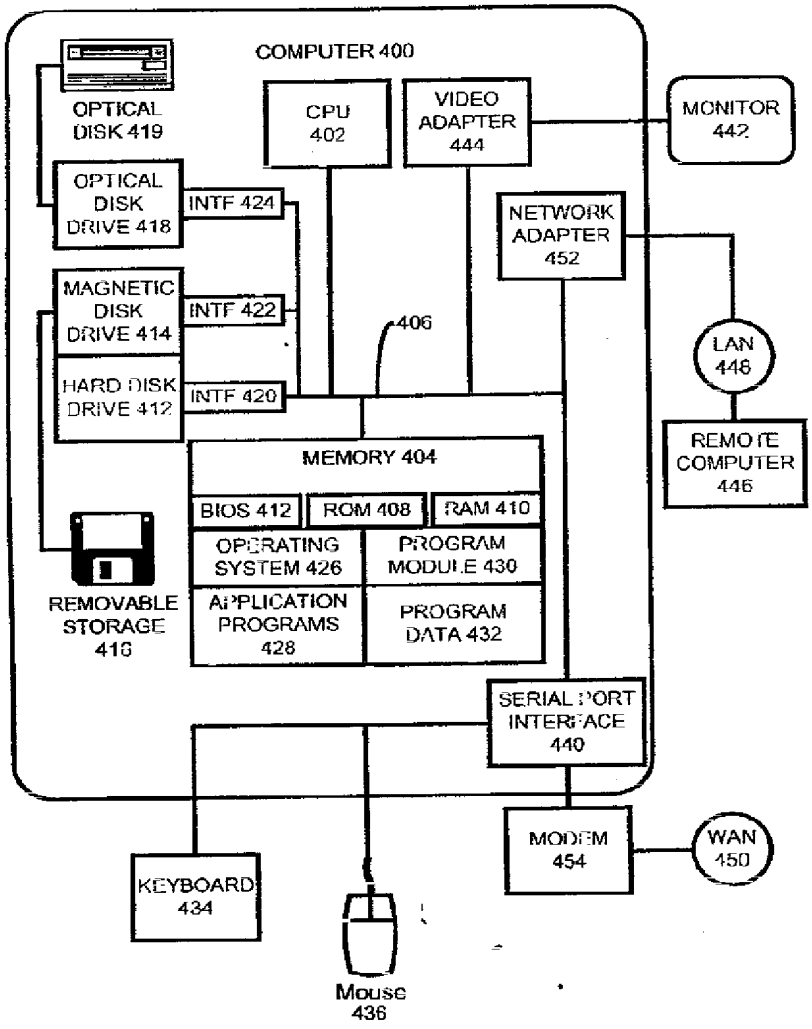
【図２】


Client
Server
Transmit an HTTP request to server with a URL specifying an ASP+ Resource
200
Receive the HTTP request, including the transportable state structure
202
Read the ASP+ Resource
203
Generate a Control Object hierarchy for the specified ASP+ Resource
204
Load the viewstate
205
Process client-side user interface element using server-side Control Objects
206
Save the viewstate
209
Render HTML data for client-side user-interface elements
208
214
Receive the HTTP response with HTML data for new page
210
Transmit HTML to the Client in a HTTP response
Display the new page in the browser with HTML data describing the client-side user interface elements
216
212
Dispose of the Control Object Hierarchy

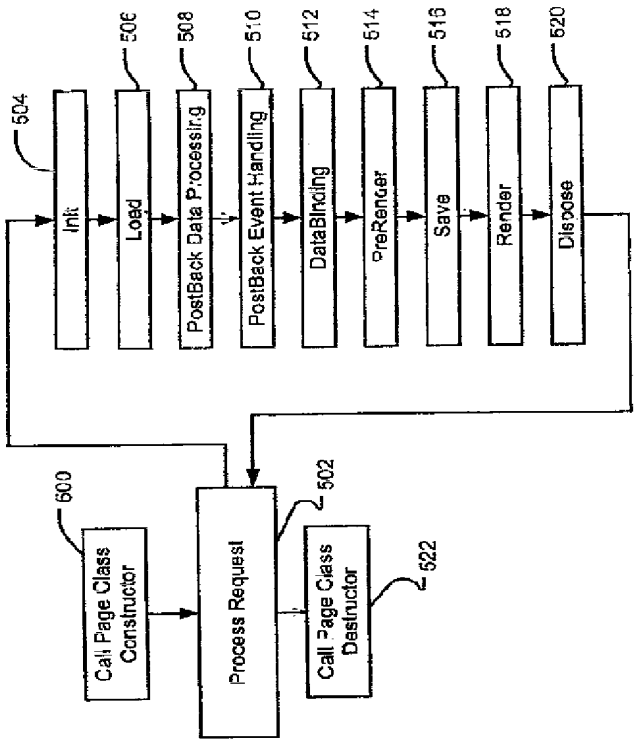
【図３】


Web Server
300
302
HTTP
Pipeline
304
Page Factory
308
ASP+
Resource
310
Control
Class
Library
312
306
Page
Object
314
330
Non-UI
Server
Components
Text Box
Object
Button
Object
Button
Object
318
320
322
324
316

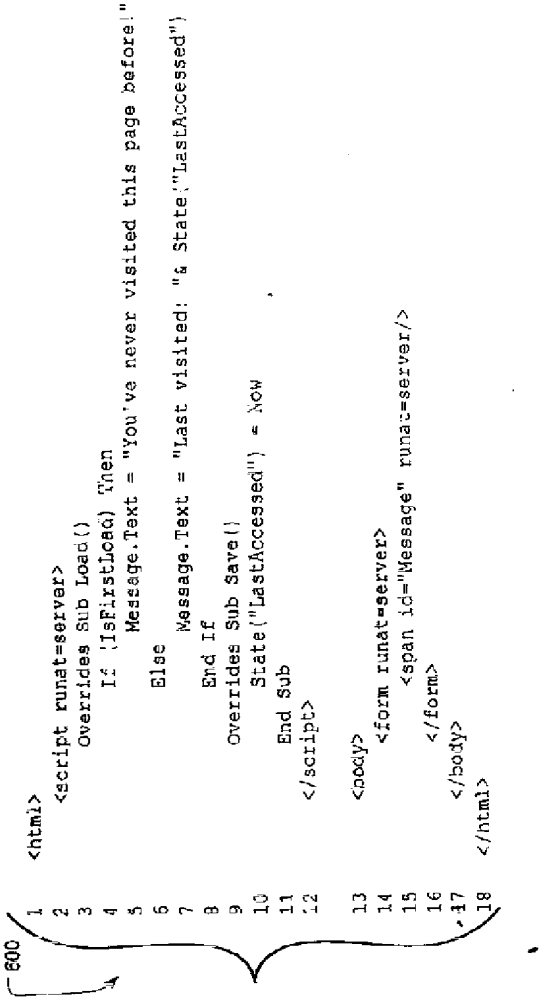
【図4】


COMPUTER 400
OPTICAL DISK 419
CPU 402
VIDEO ADAPTER 444
MONITOR 442
OPTICAL DISK DRIVE 418
INTF 424
NETWORK ADAPTER 452
MAGNETIC DISK DRIVE 414
INTF 422
406
LAN 448
HARD DISK DRIVE 412
INTF 420
REMOTE COMPUTER 446
MEMORY 404
BIOS 412
ROM 408
RAM 410
OPERATING SYSTEM 426
PROGRAM MODULE 430
REMOVABLE STORAGE 416
APPLICATION PROGRAMS 428
PROGRAM DATA 432
SERIAL PORT INTERFACE 440
MODEM 454
WAN 450
KEYBOARD 434
Mouse 436

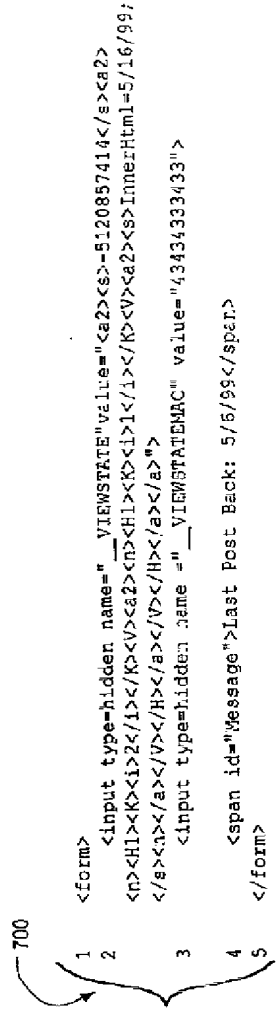
【図５】


504
Init
506
Load
508
PostBack Data Processing
510
PostBack Event Handling
512
DataBinding
514
PreRender
516
Save
518
Render
520
Dispose
600
Call Page Class Constructor
502
Process Request
522
Call Page Class Destructor

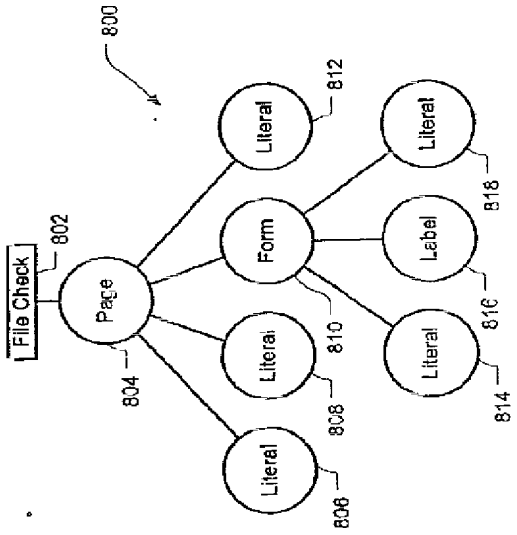
【図6】


600
1 <html>
2 <script runat=server>
3 Overrides Sub Load()
4 If (IsFirstLoad) Then
5 Message.Text = "You've never visited this page before!"
6 Else
7 Message.Text = "Last visited: "& State("LastAccessed")
8 End If
9 Overrides Sub Save()
10 State("LastAccessed") = Now
11 End Sub
12 </script>
13 <body>
14 <form runat=server>
15
16 </form>
17 </body>
18 </html>

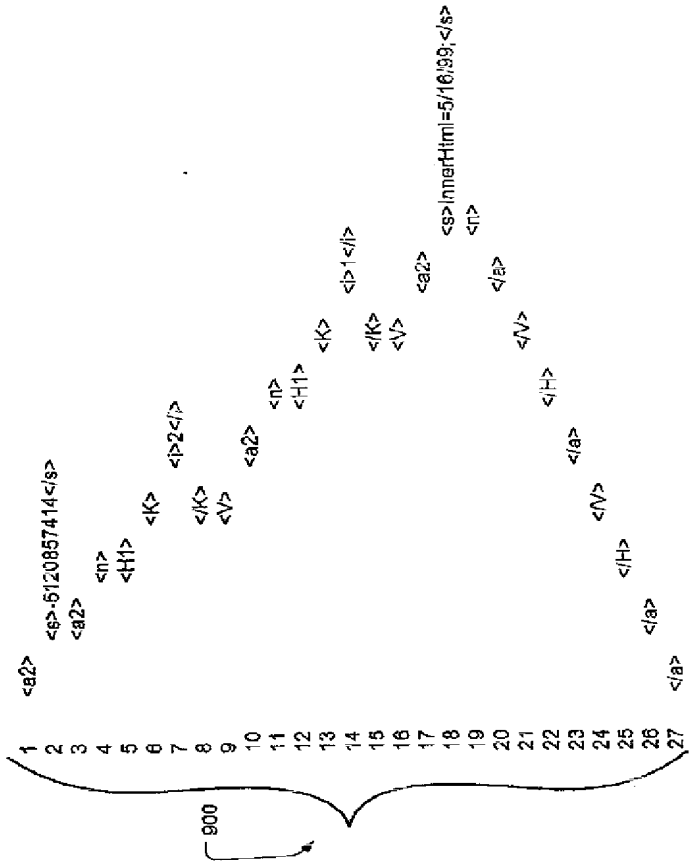
【図7】


700
1
2
3
4
5
<form>
<input type=hidden name="__VIEWSTATE"value="<a2><s>-5120857414</s><a2>
<n><H1><K>2</K><V><a2><n><H1><K>1</K><V><a2><s>InnerHtml=5/16/99;
</s><n></V></H></V></H>">
<input type=hidden name ="__VIEWSTATEMAC" value="434343334433">
Last Post Back: 5/6/99
</form>

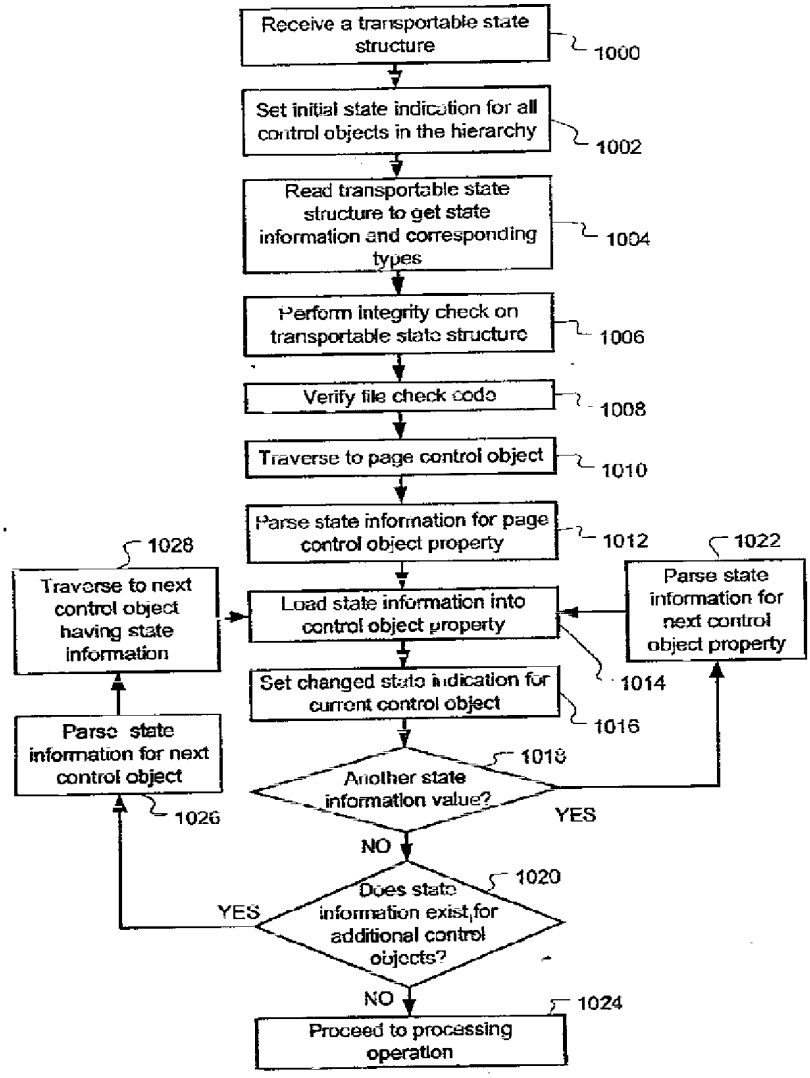
【図8】


800
File Check
802
Page
804
Literal
806
Literal
808
Literal
810
Form
812
Literal
814
Label
816
Literal
818

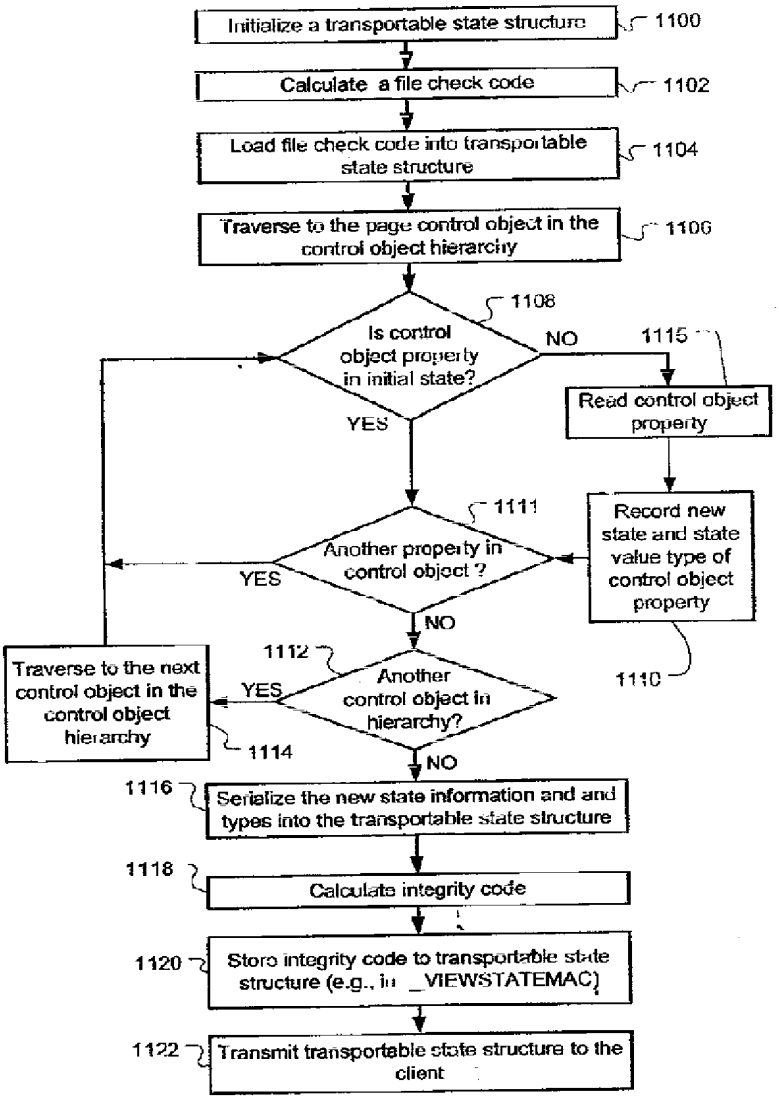
【図9】


1 <a2>
2 <s>-512085741 4</s>
3 <a2>
4 <n>
5 <H1>
6 <K>
7 2
8 </K>
9 <V>
10 <a2>
11 <n>
12 <H1>
13 <K>
14 1
15 </K>
16 <V>
17 <a2>
18 <s>InnerHtml=5/16/39;</s>
19 <n>
20
21 </V>
22 </H>
23
24 </V>
25 </H>
26
27
900

【図１０】


Receive a transportable state structure
1000
Set initial state indication for all control objects in the hierarchy
1002
Read transportable state structure to get state information and corresponding types
1004
Perform integrity check on transportable state structure
1006
Verify file check code
1008
Traverse to page control object
1010
Parse state information for page control object property
1012
1022
Parse state information for next control object property
1028
Traverse to next control object having state information
Load state information into control object property
1014
Set changed state indication for current control object
1016
Parse state information for next control object
1026
1018
Another state information value?
YES
NO
1020
Does state information exist for additional control objects?
YES
NO
1024
Proceed to processing operation

【図１１】


Initialize a transportable state structure
1100
Calculate a file check code
1102
Load file check code into transportable state structure
1104
Traverse to the page control object in the control object hierarchy
1106
1108
Is control object property in initial state?
NO
1115
Read control object property
YES
1111
Another property in control object ?
YES
Record new state and state value type of control object property
1110
NO
1112
Another control object in hierarchy?
YES
Traverse to the next control object in the control object hierarchy
1114
NO
1116
Serialize the new state information and and types into the transportable state structure
1118
Calculate integrity code
1120
Store integrity code to transportable state structure (e.g., in _VIEWSTATEMAC)
1122
Transmit transportable state structure to the client

The state of one or more server-side control objects is managed using a transportable state structure that is communicated between a web server and a client. The transportable state structure may include state information, hierarchical information, and an integrity code. The state information is related to properties of the server-side control objects. The hierarchical information is used by the server to locate an appropriate server-side control object in the hierarchy into which associated state information is to be loaded. The integrity code allows the server to determine whether the transportable state structure was corrupted during the round-trip between the server and the client. Prior to a response to the client, the state information of one or more server-side control objects is recorded into the transportable state structure for transmission to the client in the response. The transportable state structure is then returned to the server and loaded into the server-side control objects to restore the hierarchy to its previous state.

2 Representative Drawing

Fig. 10